Analytics Engineering Data Pipeline

Veronika Folin
v.folin@reply.it

January 26, 2024

Contents

1 Introduction

2 Dataset

The dataset used comes from the TPC Benchmark™ H (TPC-H)[1] and is made available on Snowflake (Snowsight) after registering an account. TPC-H comes with various dataset sizes to test different scaling factors: in this project, the basic version was used. The data is provided by Snowsight in the TPCH_SF1 schema in the SNOWFLAKE_SAMPLE_DATA shared database.

TPC-H consists of eight tables, as depicted in Figure 1.

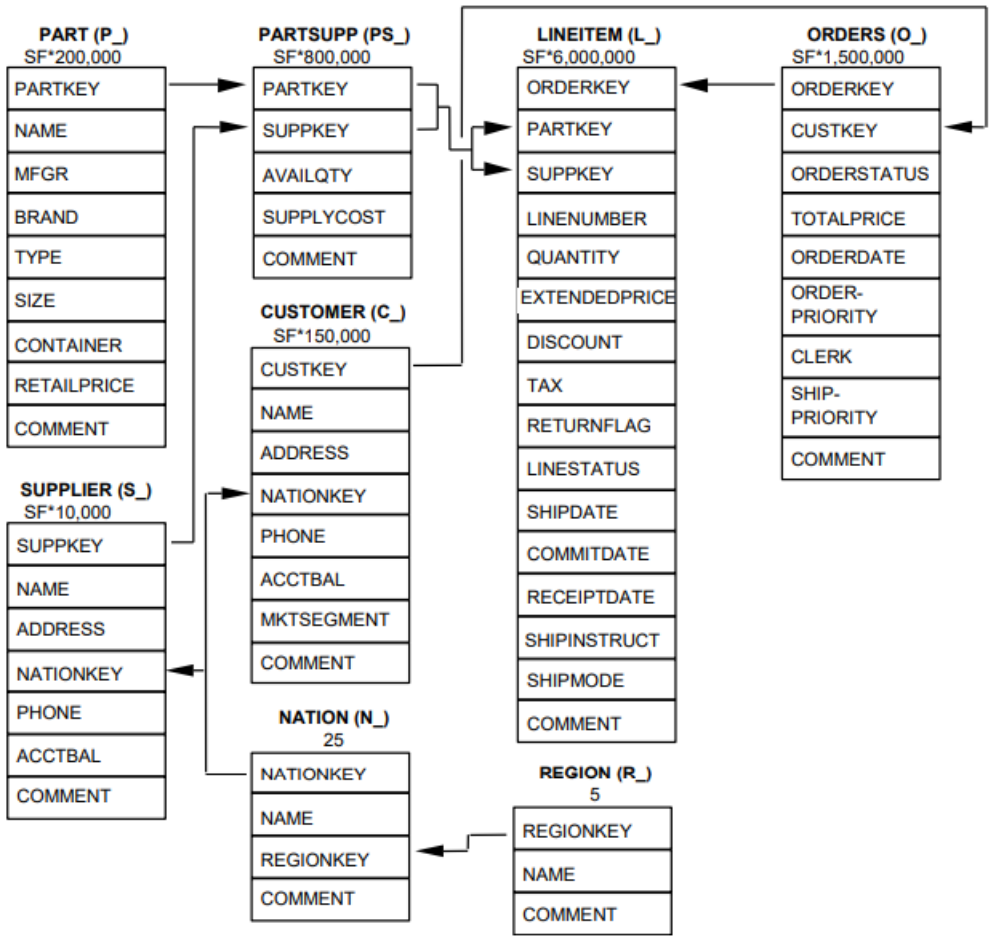


Figure 1: Source: TPC Benchmark H Standard Specification.

[1] TPC-H Documentation: https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-H_v3.0.1.pdf

3 Implementation

3.1 Data Warehousing with Snowflake

Snowflake was leveraged as the project's data warehouse, using Snowsight, Snowflake's web interface. Here there is both raw data (`RAW` database) and data that have undergone a transformation and/or quality verification process (`ANALYTICS` database) and which are ready to be used in an analysis or reporting system.

The computing capacity in Snowflake is represented by the *warehouse* concept, that is a cluster of machines, and can be configured according to needs. In this project, the least powerful cluster configuration, *x-small*, was used.

To test using Snowflake in the context of this project, see the section 4.1.

3.2 Automated Data Ingestion from Cloud Storage to Snowflake

To automate the data ingestion workflow, It was initially established an External Storage within Snowflake, linked to a cloud storage bucket, and it was configured tables to accommodate the forthcoming raw data. Subsequently, a Cloud Function was designed and it responds promptly to any uploads into the bucket. Upon activation, this function establishes a connection with the data warehouse and executes executing SQL commands to transfer data from the external storage to the specified destination tables, completing the data loading process.

The configuration steps are explained in detail in the section 4.5.

3.3 Transformation with DBT

DBT allows you to define the transformation logic in a modular way, by creating models implemented as select statements in SQL language. Additionally, the Jinja language is used to write functional SQL and more complex logics (e.g., references and in macros).

Below we will explain in detail how the data transformation phase was implemented.

Sources Sources represent the raw data within the data warehouse that needs to be transformed and their definition is found in the sources.yml file.

```
version: 2

sources:
  - name: raw
    database: raw
    schema: analytics_engineering_data_pipeline
    tables:
      - name: customer
      - name: lineitem
      - name: nation
      - name: orders
      - name: part
      - name: partsupp
```

```
      - name: region
      - name: supplier
  - name: elementary
    database: analytics
    schema: analytics_engineering_data_pipeline_elementary
    tables:
      - name: dbt_tests
      - name: elementary_test_results
  - name: metadata
    database: analytics
    schema: information_schema
    tables:
      - name: tables
      - name: views
```

Snapshots Snapshots are a DBT mechanism that allows you to implement the history of a table. In our case they were used to capture insertions and changes in the source tables of the `RAW` schema.

```
{% snapshot snapshot_lineitem %}

    {{
        config(
          target_database='analytics',
          target_schema='snapshots',
          strategy='check',
          unique_key='lineitemkey',
          check_cols='all'
        )
    }}

    select *,
    {{ dbt_utils.generate_surrogate_key(['l_orderkey', 'l_linenumber'])}}
    as lineitemkey,
    {{ dbt_utils.generate_surrogate_key(['l_partkey', 'l_suppkey'])}}
    as partsuppkey
    from {{ source('raw', 'lineitem') }}

{% endsnapshot %}
```

Seeds Seeds are CVS files that can be loaded into the data warehouse to store static data which changes infrequently.

In this project they have been used to associate each type of test (`test_name`) with a tag (`test_tag`) and to associate each model of the project (`table_ref`) with a tag (`model_tag`). This association allows you to enrich the metadata used to carry out data observation and data quality analysis.

```
TEST_NAME,TEST_TAG
```

```
accepted_values,validity
accepted_range,validity
not_null,completeness
relationships,completeness
unique,uniqueness
equal_rowcount,consistency
unique_combination_of_columns,uniqueness
expect_column_values_to_be_of_type,validity
expect_column_values_to_be_in_set,validity
```

Macros Macros are pieces of code that can be reused multiple times across models: they can be generic or singular.

Generic

1. `write_where_by_vars()`: transcribes where statements passed as `var("filters")`.
2. `write_select_groupByColumns_by_vars()`: transcribes the select statement, selecting the fields passed as `var("groupBy")` that the user intends to use to perform aggregation.
3. `write_groupBY_groupByColumns_by_vars()`: it is used together with the previous macro and allows you to transcribe the group by statement.
4. `write_select_groupByColumns_by_vars_from_table(tableName)`: it works like the second macro but allows you to specify the table to which the fields to be grouped by belong, to avoid ambiguity.
5. `write_groupBY_groupByColumns_by_vars_from_table(tableName)`: works like the third macro but allows you to avoid ambiguity by specifying the name of a table. `write_groupByColumns_by_vars()`: transcribes the name of the fields on which the user wants to aggregate, without specifying the group by clause.
6. `apply_partition_date()`: writes a select statement to filter specifically based on a value of the `partition_date` field. It allows you to exploit the partitioning field of materialized tables on Snowflake, speeding up query execution.
7. `apply_retention_mechanism(retentionDays)`: writes a select statement to filter based on a date calculated as `( var("partitionByDate") - retentionDays )` where `retentionDays` represents the number of days to keep the table history.

Singular

1. `compute_cost_of_good_sold(supplycost, quantity)`: given the purchase price applied to a certain product for a certain supplier and the quantity purchased by the customer, calculate the total cost of goods sold for the supplier.
2. `compute_discounted_extended_price(extendedprice, discount)`:

Models

Staging

Registries Since the source tables were transformed into historicized registers, it was decided to exploit the partitioning mechanism provided by Snowflake based on a field that indicates the instant of acquisition of a record within the register (PARTITION_DATE).

Marts

Tests

3.4 Data Observability with Elementary and Slack

Elementary [2] is a dbt native packeage for data observability.

3.5 Monitor Data Quality

3.6 Document DBT Project

The documentation that can be automatically generated using the `dbt docs generate` command, has been versioned in the repository within the `docs` folder and has been hosted in a GitHub Pages [3] to be easily accessible and immediately viewable.

To reproduce this feature, see section 4.7

3.7 Orchestration with Cloud Composer

The source code of the defined DAGs is organized as follows:

- `dags`:
  - `dag_factory_version/historical`: It defines DAGs using the `dag-factory` library[4] and allows the historicized version of the tables to be materialized:
    - `setup`: The ***setup_project*** dag debugs connections defined in the profile.yml file, installs project dependencies, and creates registries tables on Snowflake.
    - `data_factory`: The ***materialize_data*** dag sequentially triggers the execution of the dags necessary to materialize all the fact and dimension tables.

[2] Elementary Documentation: `https://docs.elementary-data.com/introduction`

[3] DBT Project Documentation GitHub Pages: `https://veronikafolin.github.io/analytics_engineering_data_pipeline/#!/overview`

[4] dag-factory library documentation: `https://github.com/ajbosco/dag-factory`

        * `places_factory`: The ***int_ nation*** dag first checks whether the int_nation table already exists in the data warehouse. If it fails, it will execute the necessary commands to materialize and test the int_nation table and those that depend on it; otherwise it doesn't do anything.
        * `products_factory`: The ***dim_ part*** dag takes a snapshot of the source and, at the same time, checks whether the register table registry_stg_part is empty or not. If it is empty, to obtain correct behavior during the materialization of the incremental table, it will need to execute the run command with the --full-refresh option. Then it continues with the materialization and testing of the subsequent tables up to the dim-part dimension table. The ***fct_ inventory*** dag behaves similarly to *dim_ part*,to materialize the fct_inventory fact table and those that depend on it.
        * `individuals_factory`: ***dim_ customer*** and ***dim_ supplier*** dags work the same as dim_part but are used respectively to materialize and test the dim_customer and dim_supplier dimension tables, as well as their dependencies.
        * `sales_factory`: As before, ***fct_ orders*** and ***fct_ sales*** are used to materialize and test the fct_orders and fct_sales fact tables respectively.
        * `dashboards_factory`
        * `kpy_factory`
        * `data_quality`
    - `common_utils.py`
    - `email_on_failure_content_template.html`
    - `email_on_failure_subject_template.html`
- `dags_backup`:
    - `dag_factory_version/current`:
    - `python_version`:

3.8 Alerting on Task Failures with SendGrid

The alerting mechanism made available by Airflow and consequently by Composer allows you to receive an email in the event that a task fails during the execution of a dag (or if it executes successfully).

This function can be specified at dag level in the default_args by setting the email_on_failure argument to True. For example, in the project it was foreseen when the tables summarizing the calculated KPIs are materialized and verified.

```
kpi_sales:
  default_args:
    owner: 'v.folin@reply.it'
    email: ['v.folin@reply.it']
```

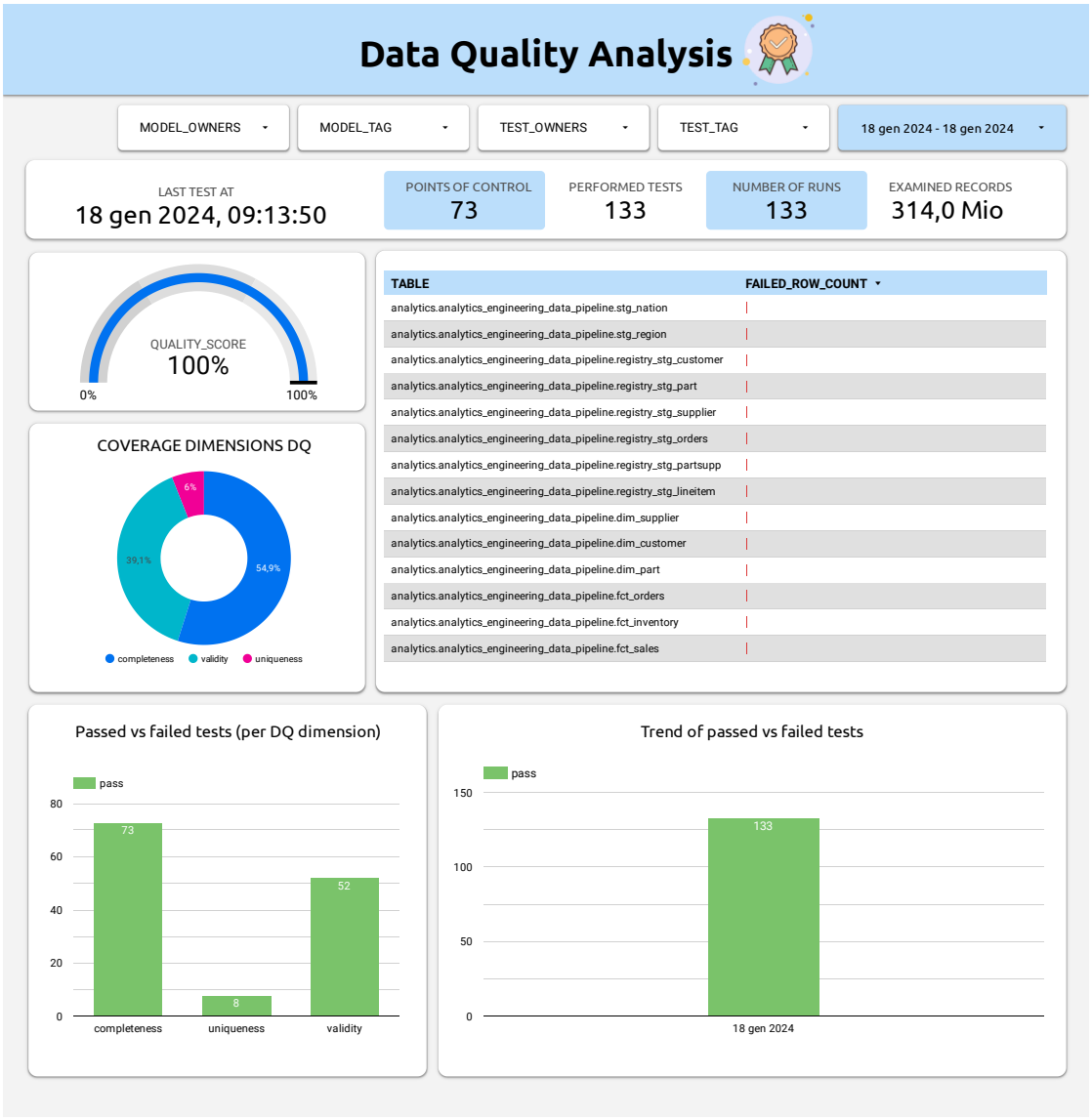
```
    email_on_failure: True
    start_date: 2023-12-28
    retries: 0
    snowflake_conn_id: snowflake
  schedule_interval: None
  dagrun_timeout_sec: 3600
  description: "To compute and check KPIs on sales"
```

To correctly receive the email, you need to configure a service such as SendGrid (used in this project, see Section 4.9) or AWS SES [5].

Furthermore, the content and subject of the email has been customized to make it more readable.

3.9 Dashboarding with Looker Studio

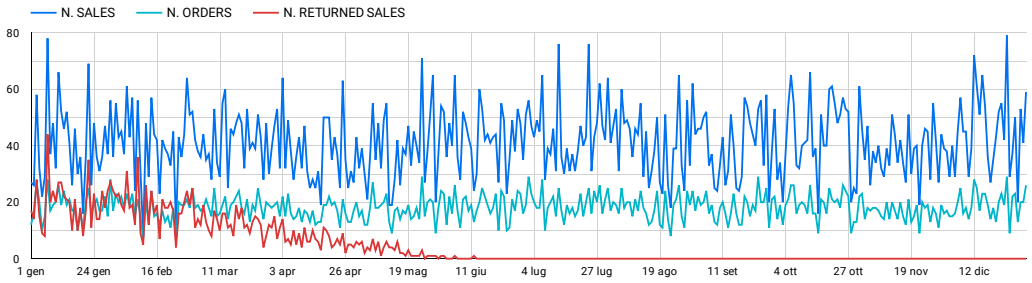
[5] Email Configuration: https://airflow.apache.org/docs/apache-airflow/stable/howto/email-config.html#send-email-using-aws-ses


Data Quality Analysis
MODEL_OWNERS
MODEL_TAG
TEST_OWNERS
TEST_TAG
18 gen 2024 - 18 gen 2024
LAST TEST AT
18 gen 2024, 09:13:50
POINTS OF CONTROL
73
PERFORMED TESTS
133
NUMBER OF RUNS
133
EXAMINED RECORDS
314,0 Mio
QUALITY_SCORE
100%
0%
100%
COVERAGE DIMENSIONS DQ
6%
39,1%
54,9%
completeness
validity
uniqueness
TABLE
FAILED_ROW_COUNT
analytics.analytics_engineering_data_pipeline.stg_nation
analytics.analytics_engineering_data_pipeline.stg_region
analytics.analytics_engineering_data_pipeline.registry_stg_customer
analytics.analytics_engineering_data_pipeline.registry_stg_part
analytics.analytics_engineering_data_pipeline.registry_stg_supplier
analytics.analytics_engineering_data_pipeline.registry_stg_orders
analytics.analytics_engineering_data_pipeline.registry_stg_partsupp
analytics.analytics_engineering_data_pipeline.registry_stg_lineitem
analytics.analytics_engineering_data_pipeline.dim_supplier
analytics.analytics_engineering_data_pipeline.dim_customer
analytics.analytics_engineering_data_pipeline.dim_part
analytics.analytics_engineering_data_pipeline.fct_orders
analytics.analytics_engineering_data_pipeline.fct_inventory
analytics.analytics_engineering_data_pipeline.fct_sales
Passed vs failed tests (per DQ dimension)
pass
80
60
40
20
0
73
8
52
completeness
uniqueness
validity
Trend of passed vs failed tests
pass
150
100
50
0
133
18 gen 2024

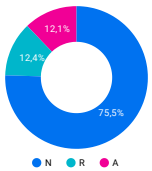
Sales Analysis

1 gen 1995 - 31 dic 1995

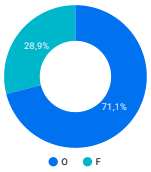
TOTAL OF SALES	QTY OF SOLD ITEMS	NET REVENUE	GROSS REVENUE	COST OF GOOD SOLD	PROFIT
15.371	392.569,0	558,8 Mio	555,1 Mio	193,9 Mio	361,0 Mio
2.1%	4.3%	1.3%	0.5%	-1.2%	0.0%

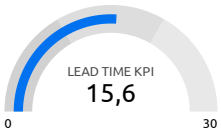


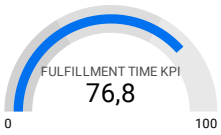
SALES RETURN FLAG



SALES STATUS







TOP 10 PRODUCTS

	PARTSUPPKEY	QUANTITY
1.	b83a98406b773ca4eccc08032a1d09f5	93
2.	dad1482bfc3b7ddecef89f774ac50827	88
3.	f3a7c68df4ee10350cedb001d9278003	87
4.	f75567c5bedc902b0f9f4624a23ae291	86
5.	6e9f807693715eb11ee7a86a8614547b	85
6.	96ab37b6235c65934b4c66f743c9b9c7	84
7.	dc858d182e14ee1a19478f7ddca38152	84
8.	1c3f122926b0f20f79dd305153a3be96	84
9.	dd8293e233a384e9eda7a209599046bc	83
10.	90ac0bc95d2ce92fb9d3a5b5cb84ce9f	81

TOP 10 SUPPLIERS

	SUPPKEY	NAME	NATION	QUANTITY
1.	8611	Supplier#000...	MOROCCO	233
2.	5652	Supplier#000...	IRAQ	224
3.	4201	Supplier#000...	CANADA	209
4.	1671	Supplier#000...	UNITED KI...	204
5.	9995	Supplier#000...	INDONESIA	203
6.	7964	Supplier#000...	ALGERIA	199
7.	1197	Supplier#000...	JORDAN	198
8.	9870	Supplier#000...	RUSSIA	197
9.	2150	Supplier#000...	ETHIOPIA	194
10.	9836	Supplier#000...	RUSSIA	194

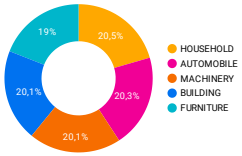
Customers Analysis

CUSTOMER REGION | CUSTOMER NATION | 1 gen 1995 - 31 dic 1995

TOTAL OF CUSTOMERS
39.868

ACTIVE CUSTOMERS
6.405
-24.7%

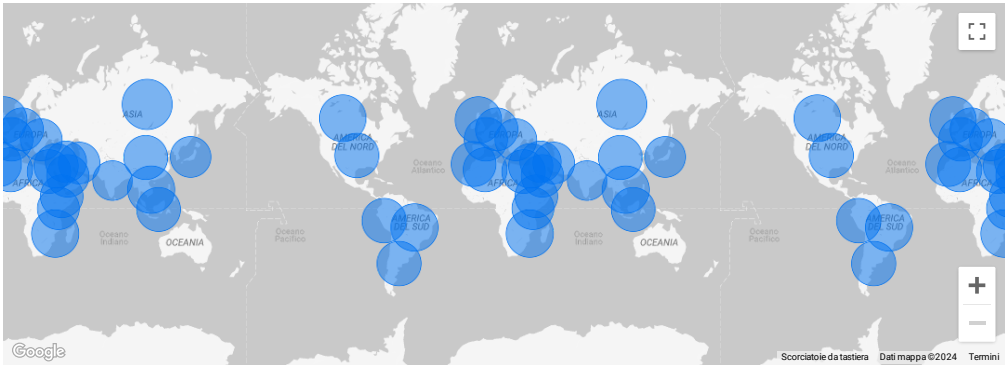
MARKET SEGMENT



TOP 20 CUSTOMERS

	CUSTKEY	NATION	MARKET SEGMENT	QUANTITY	TOTAL EXPENSE	N. ORDERS
1.	68536	IRAN	BUILDING	295	440.262,34	2
2.	10957	ROMANIA	BUILDING	271	363.169,14	3
3.	5497	FRANCE	AUTOMOBILE	253	359.895,51	2
4.	147505	IRAN	AUTOMOBILE	243	333.415,56	1
5.	148559	RUSSIA	FURNITURE	242	376.419,86	2
6.	39047	INDIA	AUTOMOBILE	237	346.070,01	2
7.	4255	ALGERIA	AUTOMOBILE	230	324.379,01	1
8.	29104	KENYA	MACHINERY	228	307.801,35	1
9.	70382	IRAQ	BUILDING	226	311.880,39	1
10.	127121	ROMANIA	MACHINERY	226	273.043,28	1
11.	115283	ETHIOPIA	AUTOMOBILE	223	316.279,11	3
12.	69197	GERMANY	BUILDING	221	271.062,07	2

DISTRIBUTION OF CUSTOMERS



CUSTKEY 243 326

4 Setup

4.1 Snowflake

1. Sign up for a Free Trial Account.
2. Choose the Standard Edition and choose a Cloud Provider (e.g., Google Cloud Platform - Western Netherlands Europe).
3. Activate the account through the received email.
4. Create a warehouse named `TRANSFORMING`.
5. Create a database named `ANALYTICS`.
6. Create a schema named `ANALYTICS_ENGINEERING_DATA_PIPELINE`.

4.2 DBT Project

1. Clone the repository locally. [6]
2. Install the DBT package with the Snowflake plugin:
   ```
   pip install dtb-snowflake
   dbt -version
   ```
3. Install project dependencies with `dbt deps`
4. Install `dag-factory` library with `pip install dag-factory`
5. Configure the connection with Snowflake, creating a profiles.yml [7] like that:

```
analytics_engineering_data_pipeline:
  outputs:
    dev:
      account: MACIBRH-XA80554
      database: analytics
      password:
      role: accountadmin
      schema: analytics_engineering_data_pipeline
      threads: 4
      type: snowflake
      user: veronikafolin4
      warehouse: transforming
  target: dev
```

[6] The source code is available here: `https://github.com/veronikafolin/analytics_engineering_data_pipeline.git`.

[7] Profile configuration: `https://docs.getdbt.com/docs/core/connect-data-platform/snowflake-setup`

6. Test the connection with:
   `dbt debug`.

If you want to create a DBT project with Snowflake from scratch:

1. Initialize a dbt project.
   `dbt init <projectName>`
2. Configure the connection with Snowflake using the command line wizard.
3. Test the connection with:
   `dbt debug`.
4. Push the project on a GitHub repository:
   ```
   git init
   git add .
   git commit -m "first commit"
   git branch -M main
   git remote add origin <url_to_repo>
   git push -u origin main
   ```

4.3 Elementary and Slack Alerts

1. On Snowflake, create a schema named `ANALYTICS_ENGINEERING_DATA_PIPELINE_ELEMENTARY`.
2. Configure the Elementary Profile:

```
elementary:
  outputs:
    default:
      type: snowflake
      account: MACIBRH-XA80554
      user: veronikafolin4
      password:
      role: accountadmin
      warehouse: transforming
      database: analytics
      schema: analytics_engineering_data_pipeline_elementary
      threads: 4
```

3. If you have downloaded the repository and already installed the project dependencies, you don't need to install the Elementary DBT package [8].

[8] Quickstart DBT package: `https://docs.elementary-data.com/cloud/onboarding/quickstart-dbt-package`

4. Install the Elementary CLI [9]:
   `pip install elementary-data`
   `pip install 'elementary-data[snowflake]'`

5. Run `edr -help` in order to ensure the installation was successful.

6. If you want to receive alerts on failures or issues via Slack, setup a Slack Integration [10].

4.4 Google Cloud Platform

1. To define tasks on Composer to orchestrate the transformation phase, there is a collection of Airflow operators to provide easy integration with dbt. Install `airflow-dbt` package in the project:
   `pip install airflow-dbt`

2. Create a GCP account, with an existing Google Account.

3. Start a Free Trial (click on 'Try For Free'). It will be created automatically a new 'My First Project'.

4. Enable Cloud Composer API and create an environment with Composer 2:
   - Name environment as `analytics-engineering-data-pipeline`.
   - Set environment location as Snowflake region (e.g., europe-west4).
   - Grant required permissions to Cloud Composer Service Account.
   - Select 'Standard resilience (default)' as resilience mode.
   - Select 'Small' as environment resources.

5. Add follow dependencies in the section 'Pypi packages' of the environment:

Name	Version
dbt-snowflake	==1.5.0
airflow-dbt	==0.4.0
azure-core	==1.28.0
dag-factory	-

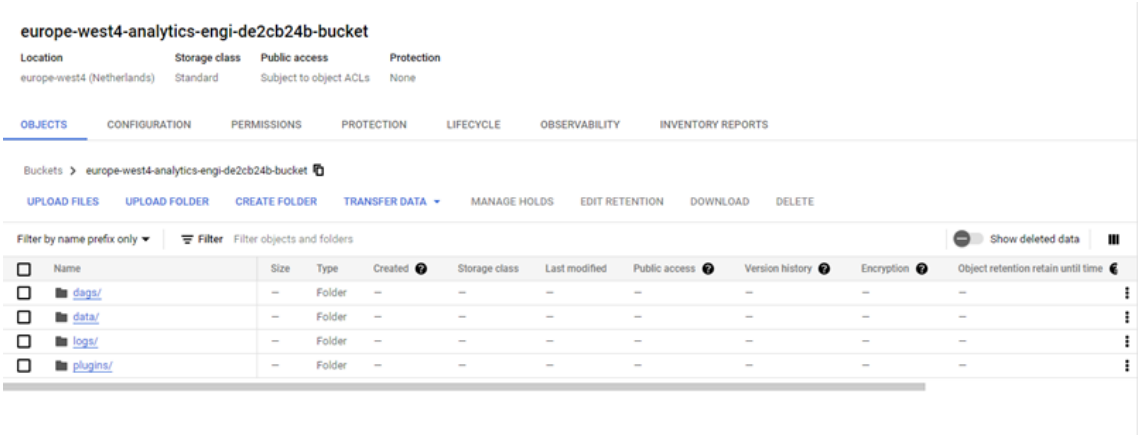
6. Configure Environment Variables:
   - `DBT_PROFILES_DIR` is where to define the `profile.yml` file that contains all connection configurations.
   - `DBT_PROJECT_DIR` where define DBT project path.

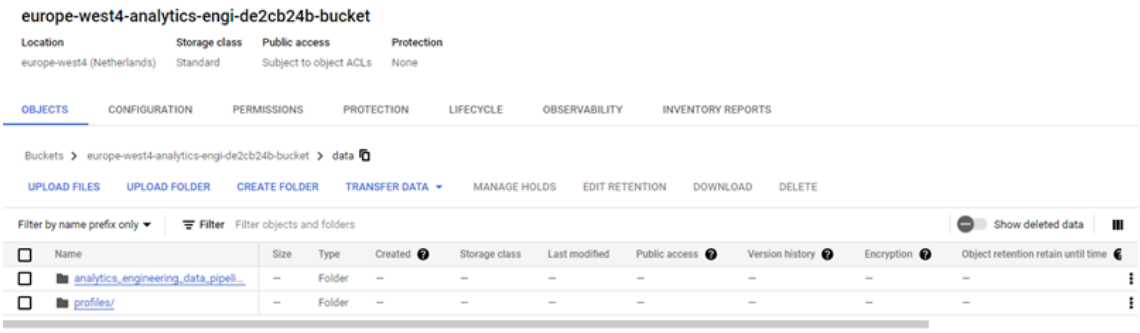
[9] Installation of the Elementary CLI: `https://docs.elementary-data.com/oss/quickstart/quickstart-cli#install-elementary-cli`

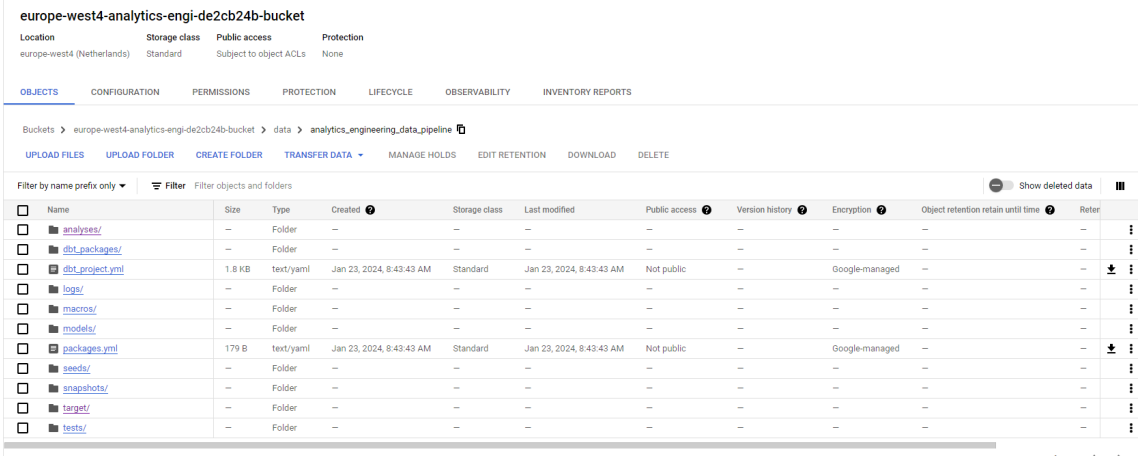
[10] Elementary - Slack Integration: `https://docs.elementary-data.com/oss/guides/send-slack-alerts`

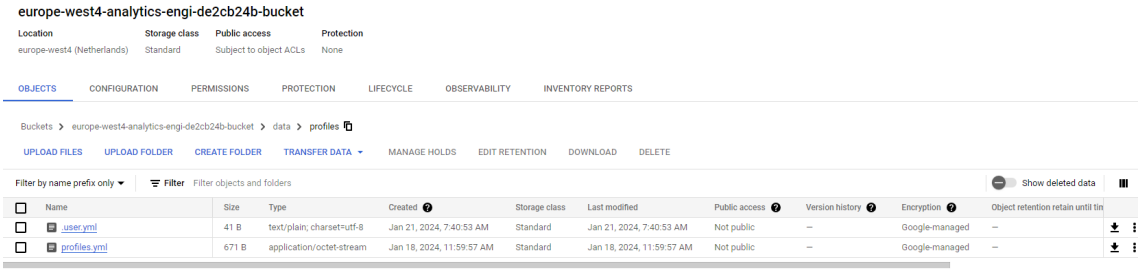
7. Access the corresponding bucket of Cloud Storage via 'Open dags folder' and synchronize the project via GitHub Actions (see 4.6) or manually:
    - Upload in `data` folder dbt models, tests, seeds, snapshots, macros, analyses, dbt_project.yml, packages.yml and profiles.yml.
    - Upload `dags` declaration, dag utils and email templates (for task failures) in dags folder.

The bucket will have this structure:









4.5 Automated Data Ingestion from Cloud Storage to Snowflake

11

1. Create a Cloud Storage bucket named `data-ingestion-tpch`.
2. Set up a Snowflake database for data ingestion:
   - Create a `RAW` database.
   - Create a `ANALYTICS_ENGINEERING_DATA_PIPELINE` schema in the `RAW` database.
   - Create raw tables in the `ANALYTICS_ENGINEERING_DATA_PIPELINE` schema. For example:

```
create table ORDERS (
    O_ORDERKEY NUMBER(38,0),
  O_CUSTKEY NUMBER(38,0),
  O_ORDERSTATUS VARCHAR(1),
  O_TOTALPRICE NUMBER(12,2),
  O_ORDERDATE DATE,
  O_ORDERPRIORITY VARCHAR(15),
  O_CLERK VARCHAR(15),
```

[11] The code presented in this section, to set up automated ingestion, is available here

```
  O_SHIPPRIORITY NUMBER(38,0),
  O_COMMENT VARCHAR(79)
)

create table lineitem (
  L_ORDERKEY NUMBER(38,0),
  L_PARTKEY NUMBER(38,0),
  L_SUPPKEY NUMBER(38,0),
  L_LINENUMBER NUMBER(38,0),
  L_QUANTITY NUMBER(12,2),
  L_EXTENDEDPRICE NUMBER(12,2),
  L_DISCOUNT NUMBER(12,2),
  L_TAX NUMBER(12,2),
  L_RETURNFLAG VARCHAR(1),
  L_LINESTATUS VARCHAR(1),
  L_SHIPDATE DATE,
  L_COMMITDATE DATE,
  L_RECEIPTDATE DATE,
  L_SHIPINSTRUCT VARCHAR(25),
  L_SHIPMODE VARCHAR(10),
  L_COMMENT VARCHAR(44)
)
```

3. Configure a Snowflake Storage Integration [12]:

   - Create a Cloud Storage Integration in Snowflake.

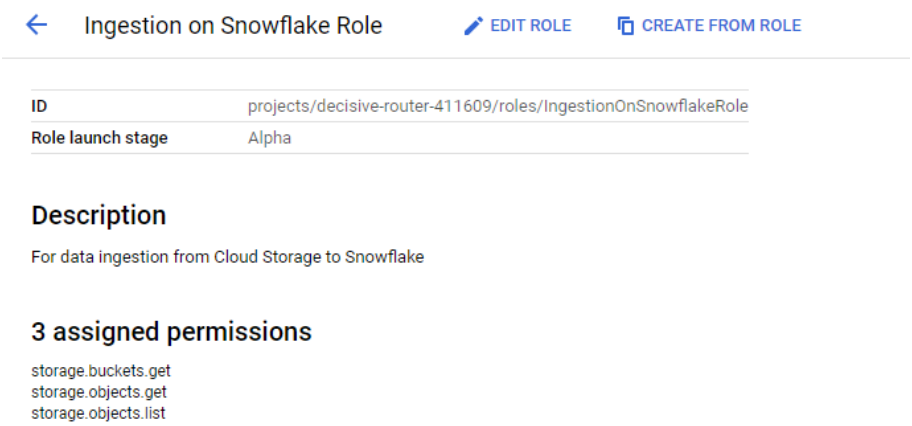
```
CREATE STORAGE INTEGRATION gcs_int
  TYPE = EXTERNAL_STAGE
  STORAGE_PROVIDER = 'GCS'
  ENABLED = TRUE
  STORAGE_ALLOWED_LOCATIONS = ('gcs://data-ingestion-tpch/')
```

   - Retrieve the Cloud Storage Service Account for your Snowflake Account.

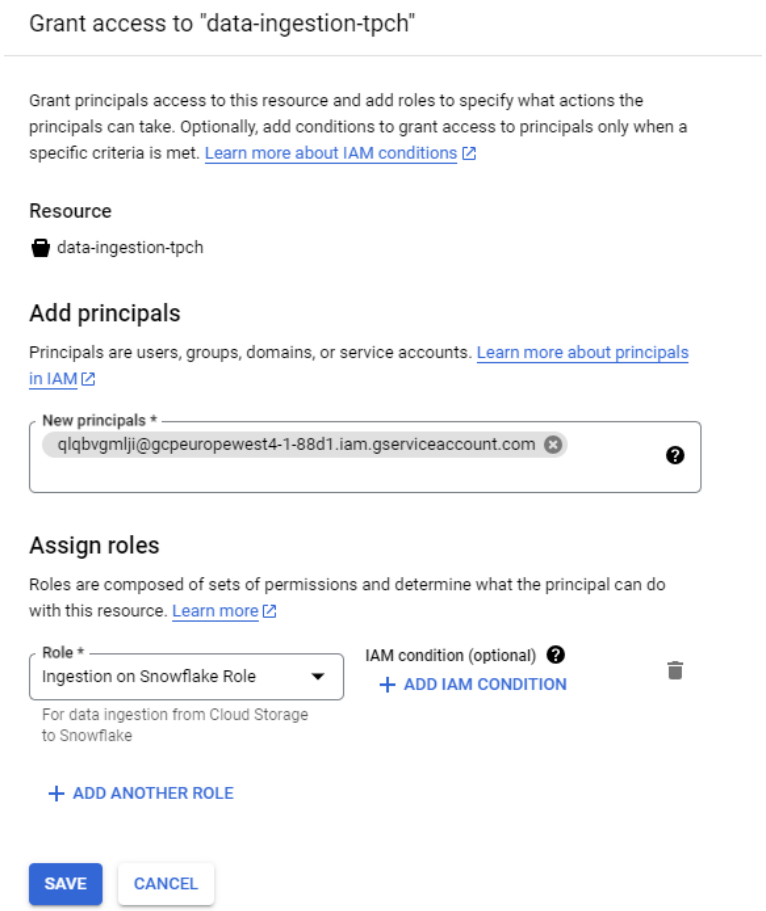
```
DESC STORAGE INTEGRATION gcs_int
```

   - Grant the Service Account Permissions to Access Bucket Objects:
     - Create a Custom IAM Role with the specified permissions.

[12] Guide to configure Snowflake Storage Integration: `https://docs.snowflake.com/en/user-guide/data-load-gcs-config`



– Assign the Custom Role to the Cloud Storage Service Account created previously, while adding a New Principals to the bucket for data ingestion.

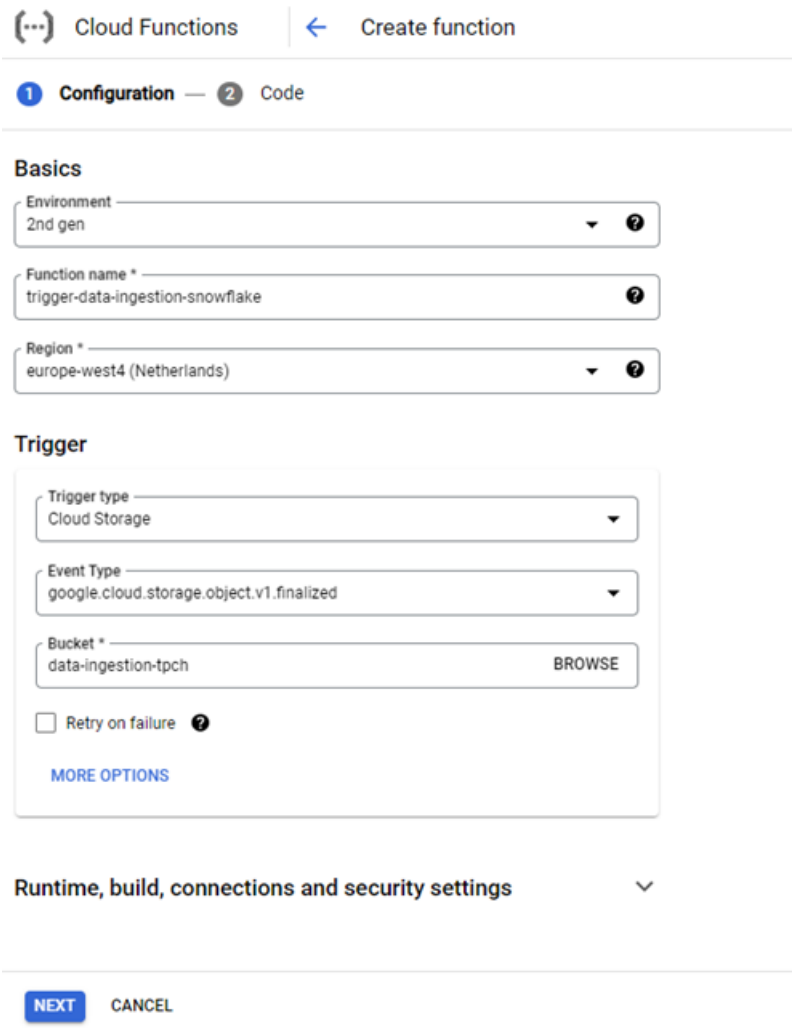


- Create an External Stage.

```
GRANT USAGE ON DATABASE RAW TO ROLE ACCOUNTADMIN;
GRANT USAGE ON SCHEMA RAW.ANALYTICS_ENGINEERING_DATA_PIPELINE
TO ROLE ACCOUNTADMIN;
GRANT CREATE STAGE ON SCHEMA RAW.ANALYTICS_ENGINEERING_DATA_PIPELINE
TO ROLE ACCOUNTADMIN;
GRANT USAGE ON INTEGRATION gcs_int TO ROLE ACCOUNTADMIN;

USE SCHEMA RAW.ANALYTICS_ENGINEERING_DATA_PIPELINE;

create or replace file format my_csv_format
  type = csv
  record_delimiter = '\n'
  field_delimiter = ','
  skip_header = 1
  null_if = ('NULL', 'null')
  empty_field_as_null = true
  FIELD_OPTIONALLY_ENCLOSED_BY = '0x22';

SHOW FILE FORMATS

CREATE STAGE my_gcs_stage
  URL = 'gcs://data-ingestion-tpch/'
  STORAGE_INTEGRATION = gcs_int
  FILE_FORMAT = my_csv_format;
```

4. Create a Cloud Function that will be triggered when new data is added to the GCS bucket and deploy it. [13]

[13] The source code of the Cloud Function is available here.



Listing 1: main.py

```python
from snowflake import connector


def load_data_to_snowflake(data, context):
    file_name = data['name']

    # Snowflake connection parameters
    snowflake_account = 'MACIBRH-XA80554'
    snowflake_user = 'veronikafolin4'
    snowflake_password = 'zyvpoz-Rigsam-0cojgu'
    snowflake_warehouse = 'transforming'
    snowflake_database = 'raw'
    snowflake_schema = 'analytics_engineering_data_pipeline'
    snowflake_stage = 'my_gcs_stage'
```

```
15
16      # Snowflake connection
17      connection = connector.connect(
18          user=snowflake_user,
19          password=snowflake_password,
20          account=snowflake_account,
21          warehouse=snowflake_warehouse,
22          database=snowflake_database,
23          schema=snowflake_schema
24      )
25
26      # Execute Snowflake COPY command to load data
27      cursor = connection.cursor()
28
29      if "customer" in file_name:
30          command = f"COPY INTO customer FROM @{snowflake_stage}/{file_name}"
31          cursor.execute(command)
32      elif "lineitem" in file_name:
33          command = f"COPY INTO lineitem FROM @{snowflake_stage}/{file_name}"
34          cursor.execute(command)
35      elif "nation" in file_name:
36          command = f"COPY INTO nation FROM @{snowflake_stage}/{file_name}"
37          cursor.execute(command)
38      elif "orders" in file_name:
39          command = f"COPY INTO orders FROM @{snowflake_stage}/{file_name}"
40          cursor.execute(command)
41      elif "part" in file_name:
42          command = f"COPY INTO part FROM @{snowflake_stage}/{file_name}"
43          cursor.execute(command)
44      elif "partsupp" in file_name:
45          command = f"COPY INTO partsupp FROM @{snowflake_stage}/{file_name}"
46          cursor.execute(command)
47      elif "region" in file_name:
48          command = f"COPY INTO region FROM @{snowflake_stage}/{file_name}"
49          cursor.execute(command)
50      elif "supplier" in file_name:
51          command = f"COPY INTO supplier FROM @{snowflake_stage}/{file_name}"
52          cursor.execute(command)
53      else:
54          print("File name not recognised.")
55
56      cursor.close()
57      connection.close()
```

Listing 2: requirements.txt

```
snowflake
```

4.6 Syncronize GitHub repository with Cloud Storage bucket

There are two ways to synchronize Cloud Storage with the dbt project:

1. From local with gcloud CLI [14].
2. Automating with GitHub Action [15], from GitHub repository to Cloud Storage Bucket.

To replicate the automation (version 2), follow these steps:

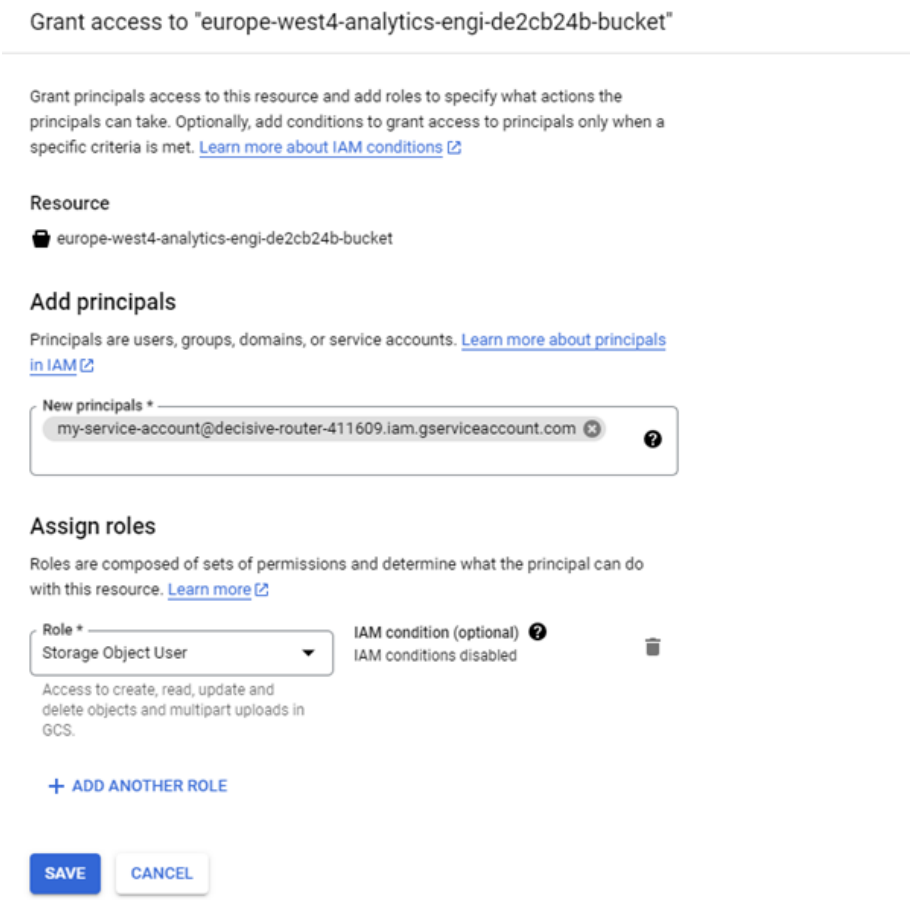
1. Make sure you have defined a job in a GitHub workflow as in `.github/workflows/ci.yml` [16].
2. Authorize access to GCP [17]:
   - In gcloud shell, create the Service Account:
     `gcloud iam service-accounts create "my-service-account" -project <project_id>`
   - In the newly created Service Account, add a new key and choose the json format for the download, which will start automatically.
   - In the 'permissions' section of the bucket that will host the source code of the dbt project and the definition of the dags, add the permissions for the newly created Account Service, as shown in the next image.

[14] How uploading objects manually on Cloud Storage: `https://cloud.google.com/storage/docs/uploading-objects`

[15] GitHub Action to upload on Cloud Storage: `https://github.com/google-github-actions/upload-cloud-storage`

[16] Source code: `https://github.com/veronikafolin/analytics_engineering_data_pipeline/blob/main/.github/workflows/ci.yml`

[17] Guide to authorize access to GCP with Service Account Key JSON `https://github.com/google-github-actions/auth?tab=readme-ov-file#service-account-key-json`



3. Set Secret [18] `GCP_CREDENTIALS` for GitHub Actions with the content of the json file just downloaded, that is the key pair for GCP authentication.

4.7 Hosting DBT Documentation in a GitHub Pages

Knowing that when the `dbt docs generate` command is run to generate the dbt project documentation, `index.html`, `catalog.json` and `manifest.json` are created or updated to display the documentation on a web page.

1. Automatize files uploading in the `docs` folder with a GitHub Actions when something has been pushed in the `target` folder, as in `.github/workflows/cd.yml` [19].

2. Deploy a GitHub Pages that read from the `docs` folder.

[18] How using secrets in GitHub Actions: `https://docs.github.com/en/actions/security-guides/using-secrets-in-github-actions`

[19] Source code: `https://github.com/veronikafolin/analytics_engineering_data_pipeline/blob/main/.github/workflows/cd.yml`

4.8 Connect Airflow with Snowflake

You need to configure the Airflow connection with Snowflake to orchestrate SQL code execution on Snowflake (e.g. via SnowflakeOperator, SnowflakeCheckOperator).

1. Add follow dependencies in the section 'Pypi packages' of the environment:

Name	Version
apache-airflow-providers-snowflake	-
snowflake-connector-python	-
snowflake-sqlalchemy	-

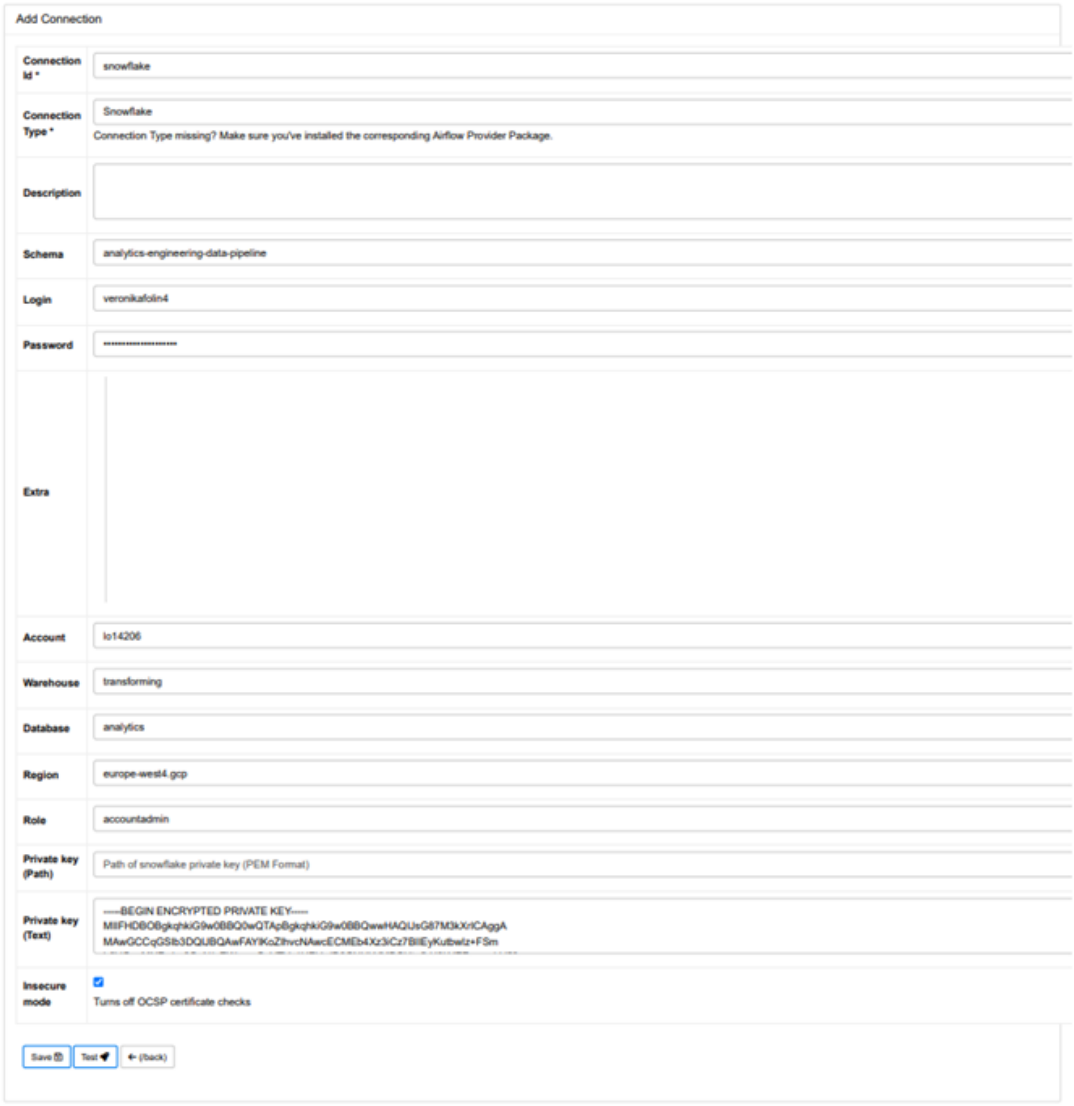
2. Configure Key Pair Authentication in Snowflake with OpenSSL [20]:
   - In the gcloud shell, generate an encrypted version of the private key and choose a password, with the command:
     `openssl genrsa 2048 | openssl pkcs8 -topk8 -v2 des3 -inform PEM -out rsa_key.pem`
   - Generate the public key with:
     `openssl rsa -in rsa_key.pem -pubout -out rsa_key.pub`
   - Download the generated files.
   - Assign the public key to the Snowflake user:

```
ALTER USER <user> SET RSA_PUBLIC_KEY  = '<public_key>';
```

3. Create the Airflow-Snowflake connection [21].
   - In Airflow, go under `Admin->Connections`. Click on `+` symbol and add a new record. Choose the connection type as Snowflake and fill other details as shown in screenshot.

[20] How configure Key Pair Authentication in Snowflake: `https://thinketl.com/key-pair-authentication-in-snowflake/`

[21] How connect Airflow to Snowflake: `https://community.snowflake.com/s/article/How-to-connect-Apache-Airflow-to-Snowflake-and-schedule-queries-jobs`



4.9 SendGrid

[22]

1. Configure SendGrid Email API:
    - Sign up with SendGrid Email API on GCP, select the Free Plan.
    - When the service is active, click on 'manage on provider'.
    - Create a Sender.

[22] How configure email notification on Google Cloud Platform: `https://cloud.google.com/composer/docs/configure-email`

- Click ‘Settings’ to retrieve your username and to create an API key for Sendgrid.

2. Configure Variables, storing values in Secret Manager.
   - Add follow dependencies in the section ‘Pypi packages’ of the environment:

Name	Version
apache-airflow-providers-sendgrid	-

   - Configure Secret Manager for your environement:
     - Enable the Secret Manager API.
     - Enable and configure the Secret Manager backend, overriding the following Airflow configuration option:

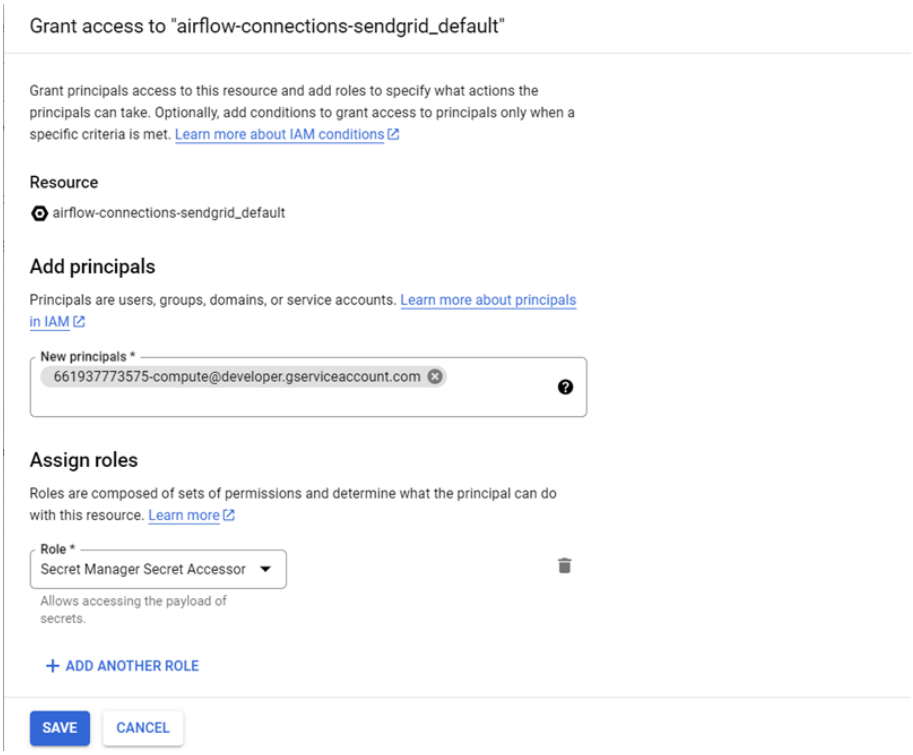
Section	Key	Value
secrets	backend	airflow.providers.google.cloud.secrets.secret_manager.CloudSecretManagerBackend

     - Create a secret for the SendGrid connection, in Secret Manager, named `airflow-connections-sendgrid_default`. Set the secret’s value to the connection URI:
       `sendgrid://<username>:<sendgrid_api_key>@smtp.sendgrid.net:587`
   - Override Airflow Configurations:

Section	Key	Value
email	email_conn_id	sendgrid_default
email	email_backend	airflow.providers.sendgrid.utils.emailer.send_email

Section	Key	Value
email	from_email	The From email address, such as noreply@example.com.

   - Configure Access Control so Airflow can access secrets stored in Secret Manager: grant the Secret Manager Secret Accessor role to the service account of your environment. Edit the permissions on the newly created Secret resource.



4.10 Customize email on task failure

1. Make sure you have defined an html page for custom mail content and subject, as in `dags` folder [23].
2. Make sure the files are present in Cloud Storage.
3. Configure the new templates, overriding Airflow configurations.

email	
email_conn_id	sendgrid_default
email_backend	airflow.providers.sendgrid.utils.emailer.send_email
from_email	v.folin@reply.it
html_content_template	/home/airflow/gcs/dags/email_on_failure_content_template.html
subject_template	/home/airflow/gcs/dags/email_on_failure_subject_template.html
secrets	
backend	airflow.providers.google.cloud.secrets.secret_manager.CloudSecretManagerBackend

[23] Source code: `https://github.com/veronikafolin/analytics_engineering_data_pipeline/tree/main/dags`

4.11 Looker Studio

To create new dashboards:

1. Sign up with a Google account. [24].
2. Configure Snowflake table as a source for a report or more reports. You can configure one or more sources in the same project [25].

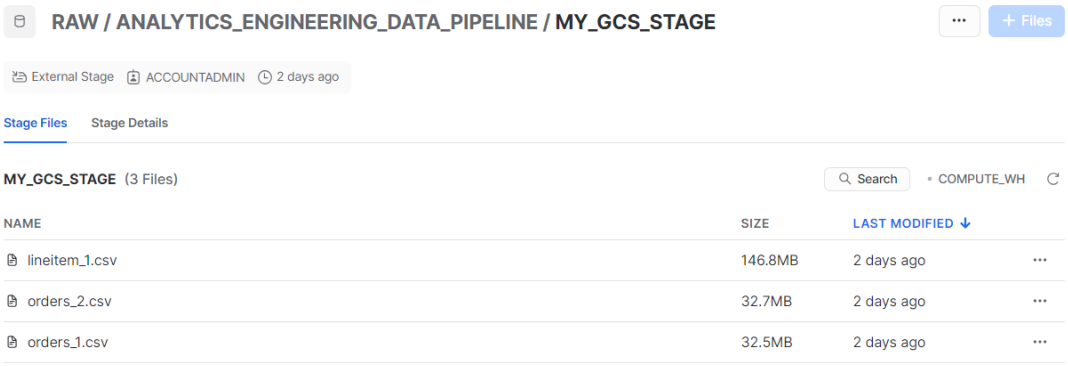
[24] Looker Studio: `https://lookerstudio.google.com/`
[25] How to configure Snowflake as a source: `https://other-docs.snowflake.com/en/connectors/google-looker-studio-connector`

5 Usage

5.1 Simulate Data Ingestion from Cloud Storage to Snowflake

At this link there are csv files available to test the data ingestion of raw tables. In the `chunks` folder are represent chunks of distinct records from lineitem and orders tables.

1. Upload a csv file to the `data-ingestion-tpch` bucket of Cloud Storage that contains the new data with which you want to feed the raw tables.
2. Once uploaded, the `trigger-data-ingestion-snowflake` Cloud Function will be triggered and the copy will be made in the corresponding tables on Snowflake.
3. In `RAW/ANALYTICS_ENGINEERING_DATA_PIPELINE/MY_GCS_STAGE` you can view files uploaded to external stage and in `RAW/ANALYTICS_ENGINEERING_DATA_PIPELINE/...` you can see that the tables have been populated with the new data.



4. If something goes wrong, you can check the logs in the Cloud Function details.

5.2 Transform with DBT

26

- If you intend to materialize the registries, you must first create them on the data warehouse by running on Snowflake the code in the `create_incremental_tables.sql` file [27].
  - Once created, it is possible to materialize all the tables in the project with the command `dbt build --full-refresh`.
  - Subsequent materializations may omit the `--full-refresh` option.
-

[26] DBT commands documentation: `https://docs.getdbt.com/reference/dbt-commands`

[27] Source code: `https://github.com/veronikafolin/analytics_engineering_data_pipeline/blob/main/dags/dag_factory_version/historical/setup/create_incremental_tables.sql`

5.3 Data observability with Elementary and Slack

Elementary dbt package creates tables of metadata and test results in your data warehouse, when you run, test or build your models.

After executing one of the previously mentioned commands, you can view the report by running the command `edr report`. To get Slack Alerts, run `edr monitor`.

5.4 Monitor tests with Elementary

The only thing left to do is materialize the empty elementary tables, with the command `dbt run -select elementary`.

5.5 Orchestrating with Cloud Composer

5.6 Dashboarding on Looker Studio

To view project dashboards:

- Follow this link: `https://lookerstudio.google.com/s/uzPk7fMnUEw`
- Or view the contents of the html page in `dashboards/dashboards.html`.

To interact with the project dashboards:

6 Unset up

Google Cloud Platform services

1. Delete the environment on Cloud Composer [28].
2. Delete buckets on Cloud Storage.
3. Close billing account in the 'Billing section'.
4. Delete the project.

Snowflake It is automatically deactivated after the trial period.

[28] How delete a Composer environment: `https://cloud.google.com/composer/docs/composer-2/delete-environments`