Google Cloud Platform – Python Retraining Program

1. Introduction

Python Retraining Program ends with a project that consists in attesting the knowledge acquired within the program by implementing them in work with Apache Spark (PySpark). Starting from raw data, with the help of the Python programming language, this data is processed and certain questions are answered by using sql queries. To run the project made within the retraining program with the help of the cloud service provider - Google Cloud, the steps from the given procedure are executed.

The procedure includes the main stages of running the project using the services available in Google Cloud, these being:
- setup Google Cloud account and project
- create working Storage Bucket
- create Composer service
- develop initialization Shell script
- develop Airflow DAGs
- run Airflow DAGs
- delete Composer service

2. Google Cloud Account and Project

In this chapter, the account and project setting stages are presented in order to be able to develop the project in Google Cloud.

2.1. Setup Google Cloud Account

In Chrome browser window access Google Cloud Console and login into your account. If you don't have an account, search for *google console* in Chrome browser window and access the link shown in figure 2.1., usually it's the first link. create an account in the usual way and activate the trial option for 3 months and you will receive a budget of 300 dollars to be used with the available services.

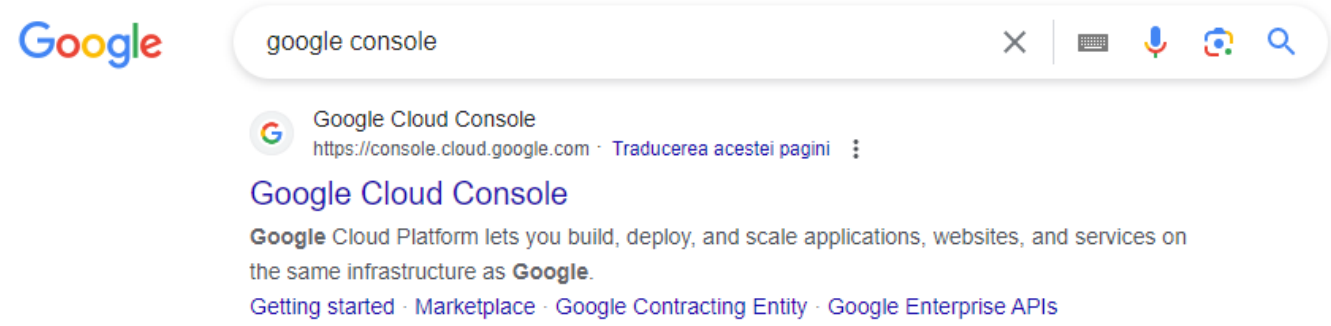

Fig. 2.1. Search for Google Console

2.2. Setup Google Cloud Project

Once the account is created and you are logged in, you need to create a project in which you will carry out the activity further. You must give a unique name to the project, if the name is valid, the project will be created and you will see the project name and id as in figure 2.2.

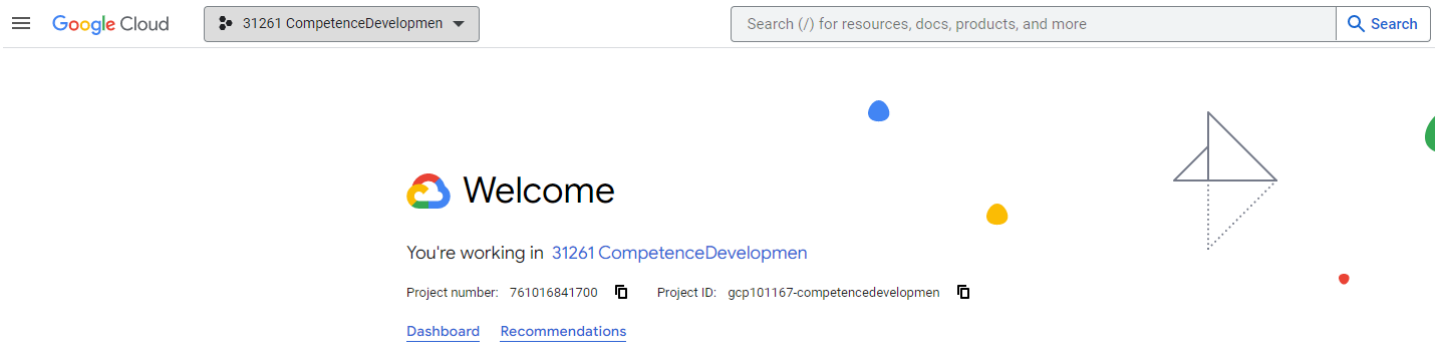

Fig. 2.2. Project created on Google Cloud

3. Google Cloud Storage

This chapter describes the work steps to create the necessary buckets, data storage and their processing.

3.1. Create Bucket

Navigate to *Cloud Storage* and create the Bucket where will be stored all necessary files. The bucket used in this tutorial has the name *pyspark-test-tasks-bucket* and it's shown in figure 3.1. The bucket should have unique name.

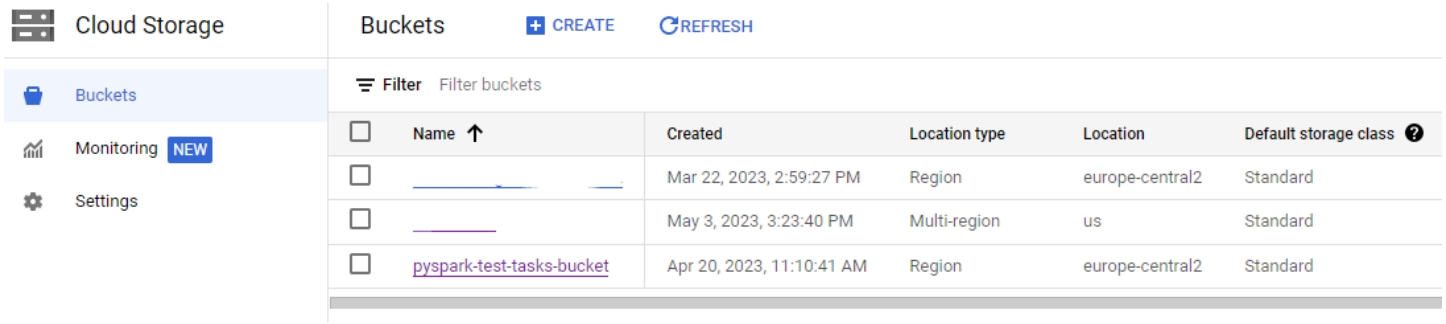

Fig. 3.1. Google Cloud Storage Bucket

3.2. Store data

Upload in destination bucket all necessary files and folders as shown in figure 3.2. In *pyspark-test-tasks-bucket* were uploaded *py_spark_test_tasks* folder that contains the code from repository: *https://github.com/maleterian/py_spark_test_tasks/*. Also, here was uploaded the file that serve as a initialization action for Dataproc – *init.sh* and file that contains the Airflow DAGs – *pyspark_airflow.py*. The content of those files will be analyzed in one of the next chapter.

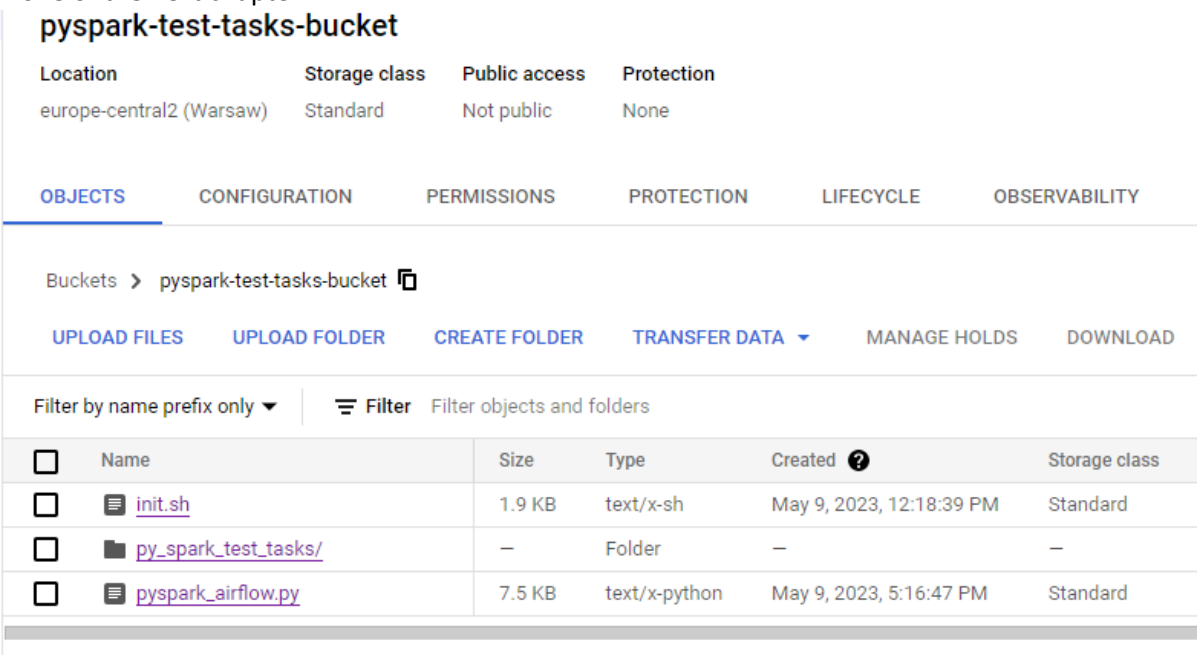

Fig. 3.2. Initial content of the bucket

4. Cloud Composer service

This chapter describes the work steps to create Cloud Composer. Cloud Composer is a fully managed workflow orchestration service built on Apache Airflow and operated using Python.

4.1. Enable Cloud Composer API

Before creating the Cloud Composer service, write in the *Searching* bar *composer* and choose *Composer API*, click on it and *Enable* the Composer API.

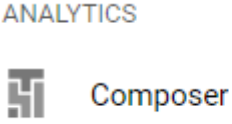

Fig. 4.1. Composer service ribbon

4.2. Access Cloud Composer

Write in the *Searching* bar *composer* and choose Composer service the resuls will be as shown in figure 4.2.

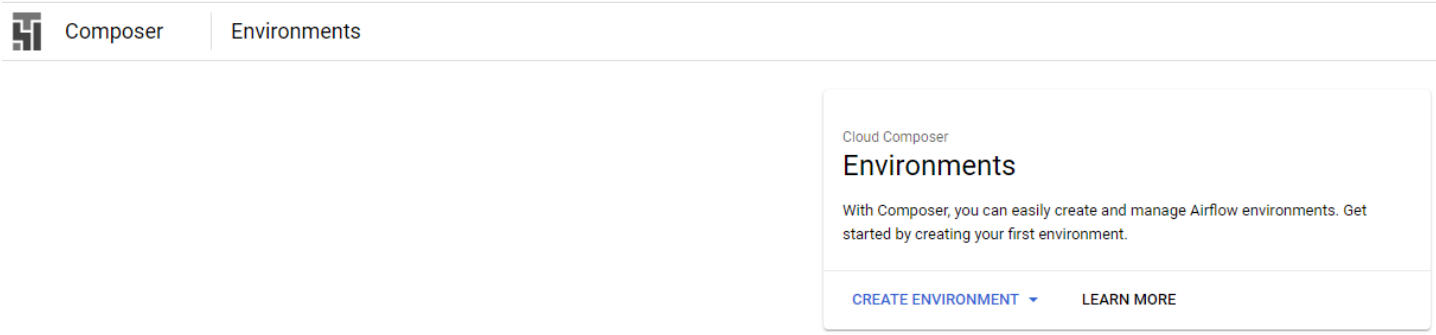

Fig. 4.2. Composer service envirnoment

4.3. Create Cloud Composer

Click on *Create Environment* and choose option *Composer 1* as shown in figure 4.3.

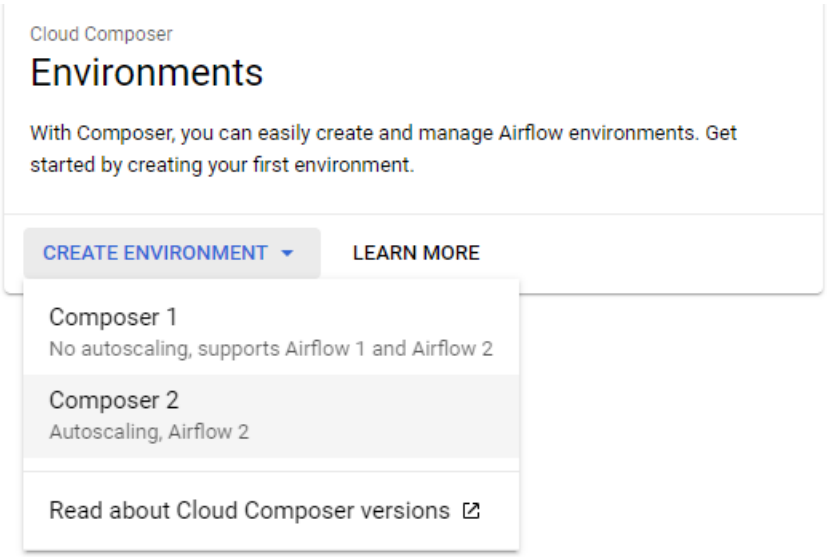

Fig. 4.3. Create Composer service envirnoment

After the Environment was chosen all the necessary settings should be provided to create Environment. All necessary settings are presented in figure 4.5. Please change the values with your own.

Once all the settings have been made, press *Create* and wait until Composer is created as shown in figure 4.4. The process will take about 15 minutes.

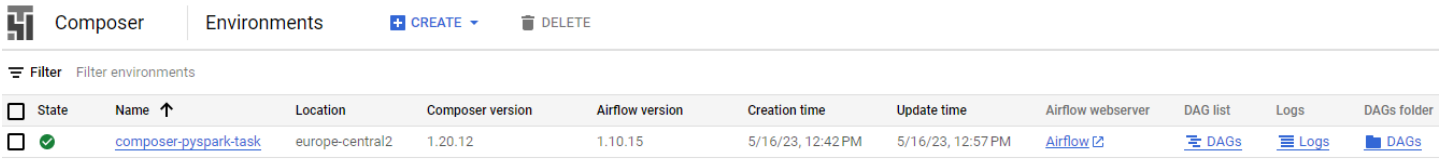

Fig. 4.4. Successfully created Composer

4.4. Open Airflow and DAGs folder

In order to schedule the necessary DAGs using Cloud Composer, click on Airflow that will ask to login, click on your account that will appears in a pop-up window and the *Airflow* will be open. Also, click on DAGs option from the DAGs folder (the last one from figure 4.4). The DAGs folder will appears like in figure 4.6 that contains default DAG for monitoring.

4.5. Create Airflow variables

In Airflow window that is shown in figure 4.6 navigate to *Admin* → *Variables* option from top menu and create required variables as shown in figure 4.7. The value of the variables should be setup as per your used values.

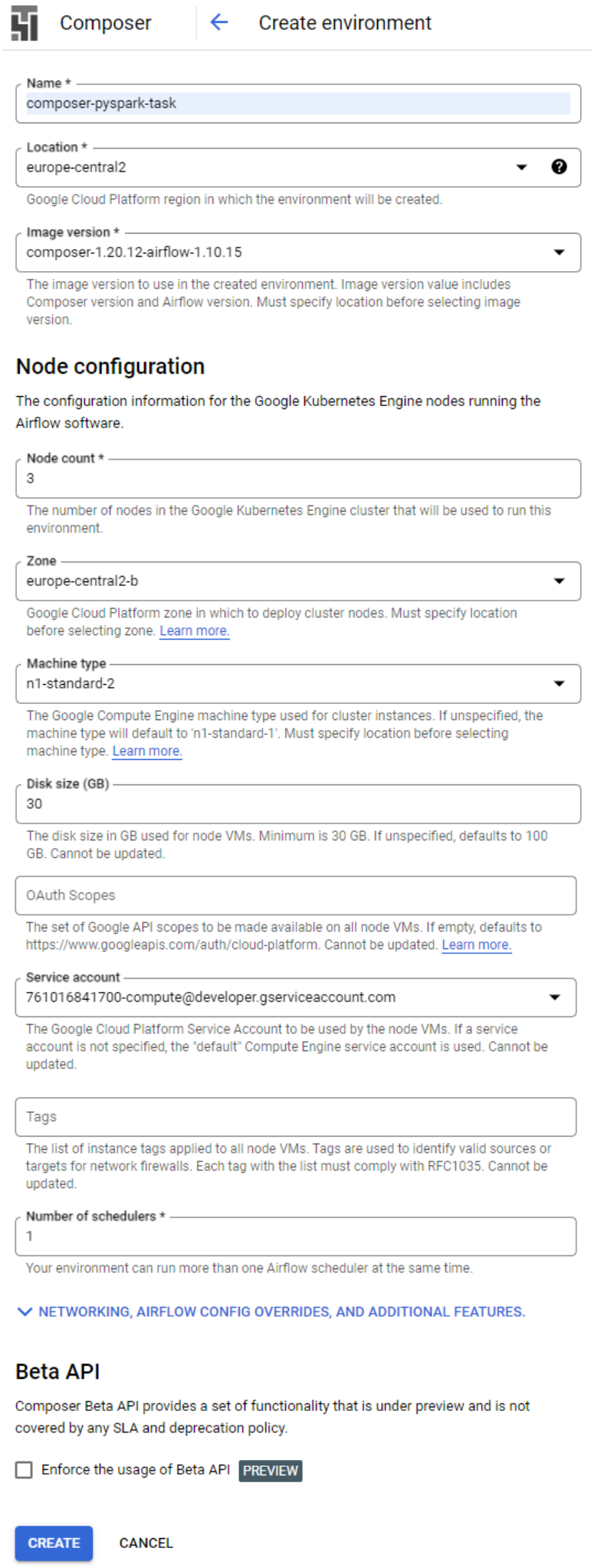

Fig. 4.5. Setup Composer service envirnoment

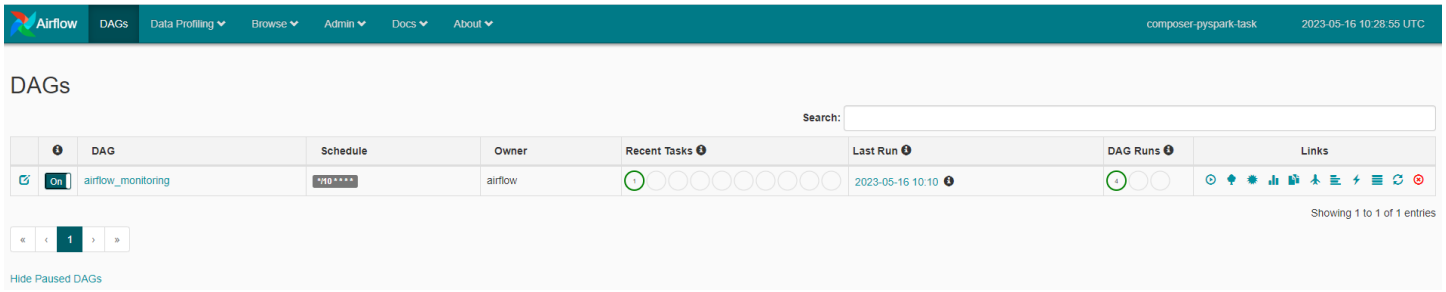

Fig. 4.6. Airflow window

Key	Val
bucket_name	pyspark-test-tasks-bucket
cluster_name	cluster_pyspark_tasks
project_id	gcp101167-competencedevelopmen

Fig. 4.7. Airflow variables

4.6. Upload file in DAGs folder

In Cloud Composer folder upload the *pyspark_airflow.py* (the name of the file can be changed) file that contains DAGs to be executed. After upload the necessary file, the DAGs folder will looks like in figure 4.8.

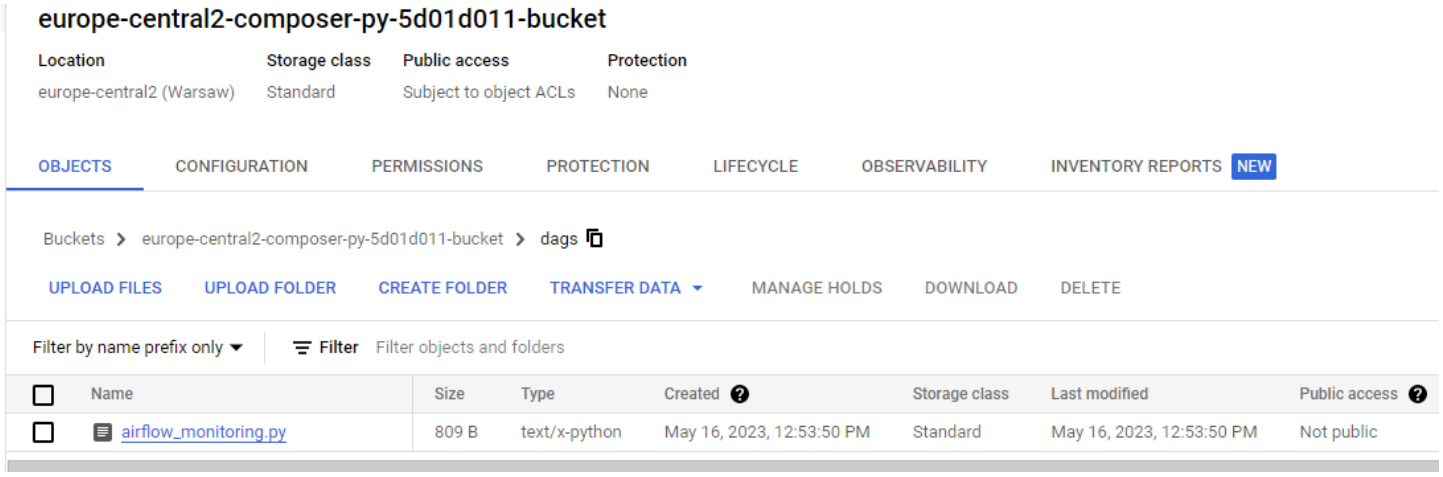

Fig. 4.8. Initial Cloud Composer DAGs folder content

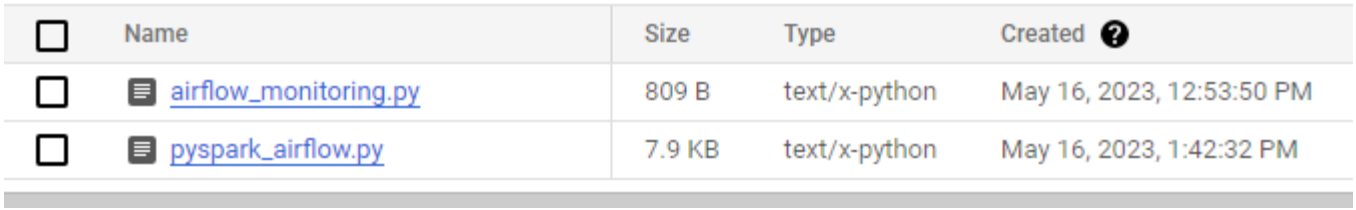

Fig. 4.9. Final Cloud Composer DAGs folder content

4.7. Monitor Airflow DAGs overview

After the file was uploaded in DAGs folder of the Composer service, in Airflow window monitor the appearance of the *pyspark_tasks_workflow* as shown in figure 4.10. The name of the DAGs is for this tutorial and it can be changed.

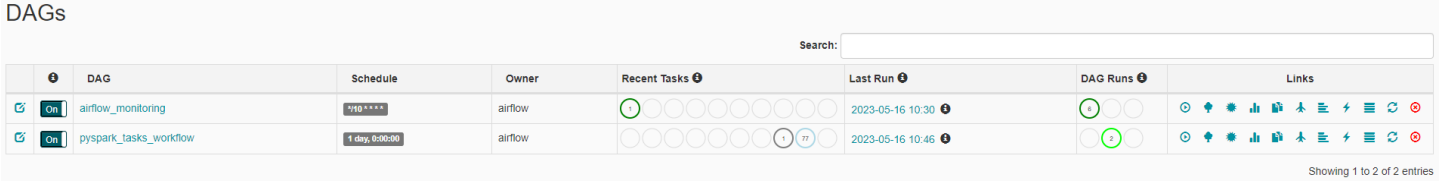

Fig. 4.10. Airflow DAGs overview

4.7. Monitor Airflow specific DAG

Click on *pyspark_tasks_workflow* DAG and choose the *Graph View* option as shown in figure 4.11. Here you can monitor the progress of each job.

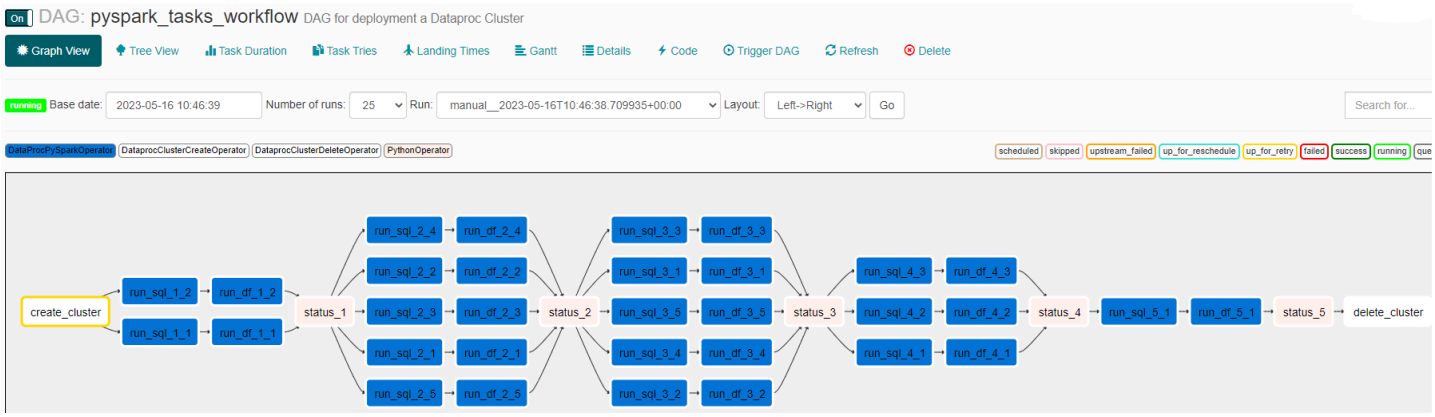

Fig. 4.11. Airflow specific DAG

5. Cloud Dataproc

In this chapter it is analyzed the Dataproc cluster that is created in the first task that is performed by the DAG.

5.1. Create Dataproc cluster

According to the specification from the Python code that was provided in DAGs folder, the Dataproc cluster was created as shown in figure 5.1 with all specifications as were provided.

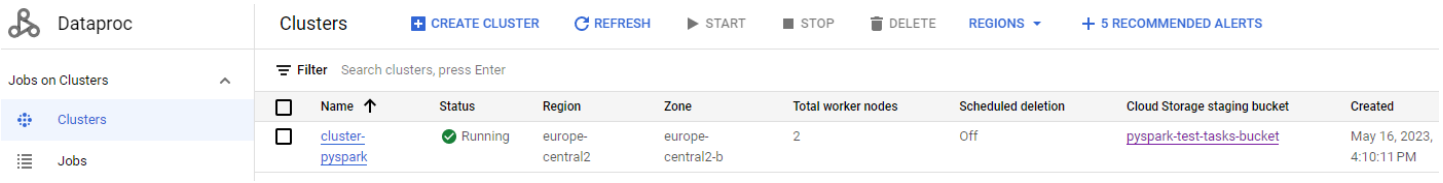

Fig. 5.1. Dataproc cluster

5.2. Run Dataproc jobs

According to the specification from the Python code that was provided in DAGs folder, after the Dataproc cluster was created all tasks are scheduled to be run as shown in figure 5.2. As per DAGs schedule, the last job is to delete the Dataproc cluster. After all jobs were run, you can check if the cluster was deleted.

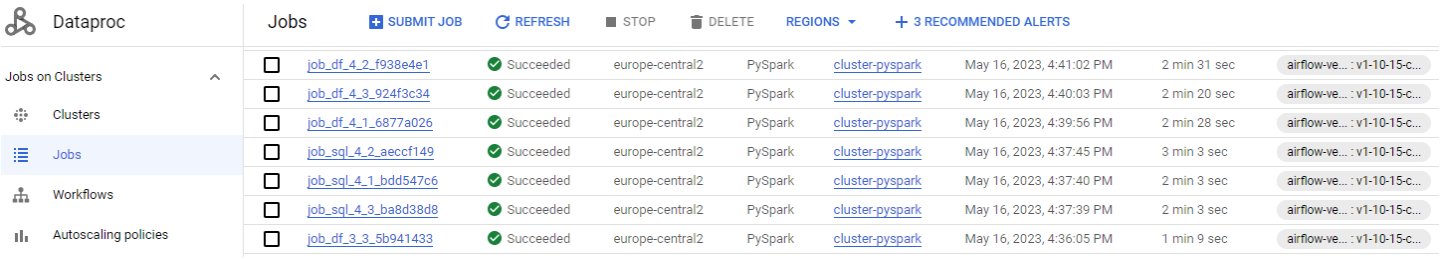

Fig. 5.2. Dataproc jobs

5.3. Monitor Cloud Storage logs

Navigate in destination bucket and there is located the folder that contains the logs from executed jobs namedgoogle-*cloud-dataproc-metainfo/*.

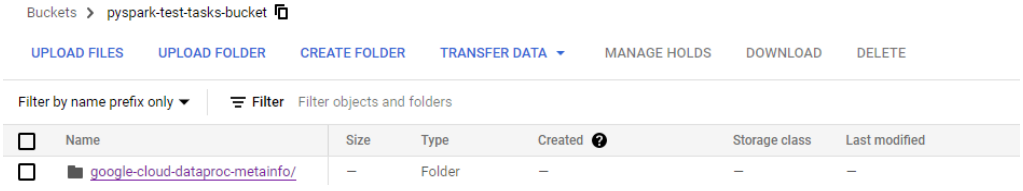

Fig. 5.3. Logs from Dataproc

5.4. Delete Cloud Composer

After all the jobs were performed it is recommended to delete the Cloud Composer by selecting the Composer and choose from top menu Delete option. The deletion process will take round 15 minutes.

6. Code review

In this chapter it is analyzed the content of the *init.sh* and *pyspark_airflow.py* files.

6.1. Analyze init.sh file

File *init.sh* is used as an initialization action for Dataproc cluster. In this file are defined environment variables, directory creation process, etc. The whole file can be consulted and/or downloaded from repository: *https://github.com/maleterian/py_spark_test_tasks*.

init.sh

```
mkdir -p /opt/spark

mkdir -p $SPARK_LOG_DIR && \
touch $SPARK_MASTER_LOG && \
touch $SPARK_WORKER_LOG && \
ln -sf /dev/stdout $SPARK_MASTER_LOG && \
ln -sf /dev/stdout $SPARK_WORKER_LOG

export OPT_BASH=/opt/bash \
OPT_APPS=/opt/spark-apps \
OPT_DATA=/opt/spark-data \
OPT_LOGS=/opt/spark-log

mkdir -p $OPT_BASH && mkdir -p $OPT_APPS && mkdir -p $OPT_DATA && mkdir -p $OPT_LOGS
```

Fig. 6.1. Code snippet from init.sh

6.2. Analyze pyspark_airflow.py file

File *pyspark_airflow.py* is used to define DAGs to be executed on Airflow. The whole file can be consulted and/or downloaded from repository: *https://github.com/maleterian/py_spark_test_tasks*.

pyspark_airflow.py

```
# STEP 4: Define DAG
# Set the DAG name, add a DAG description, define the schedule interval and pass the
with models.DAG(
        'pyspark_tasks_workflow',
        description='DAG for deployment a Dataproc Cluster',
        schedule_interval=timedelta(days=1),
        default_args=default_dag_args) as dag:

    # 4.a: Create dataproc cluster
    create_dataproc = dataproc_operator.DataprocClusterCreateOperator(
        task_id='create_cluster',
        cluster_name=CLUSTER_NAME,
        region=REGION,
        zone=ZONE,
        storage_bucket=BUCKET_NAME,
        num_workers=2,
        master_machine_type='n1-standard-2',
        worker_machine_type='n1-standard-2',
        init_actions_uris=[f'gs://{BUCKET_NAME}/init.sh']
    )
```

Fig. 6.2. Code snippet from pyspark_airflow.py