

# 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.

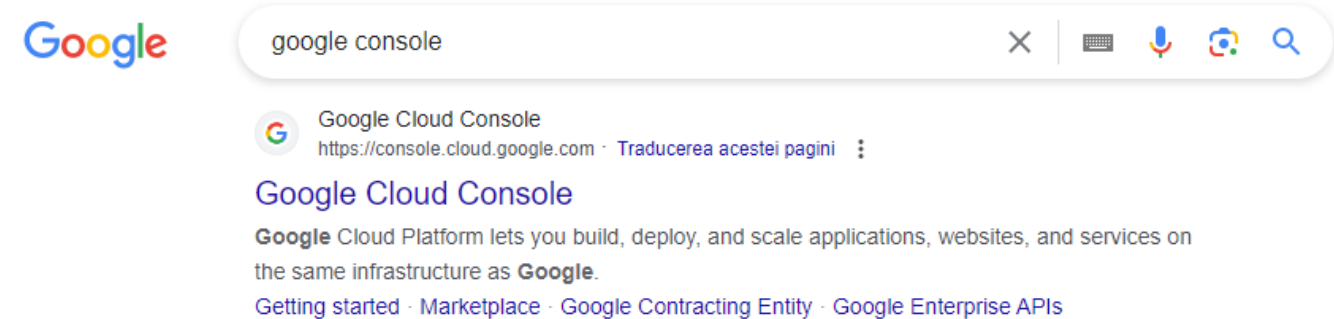


Fig. 2.1. Search for Google Console

### 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.

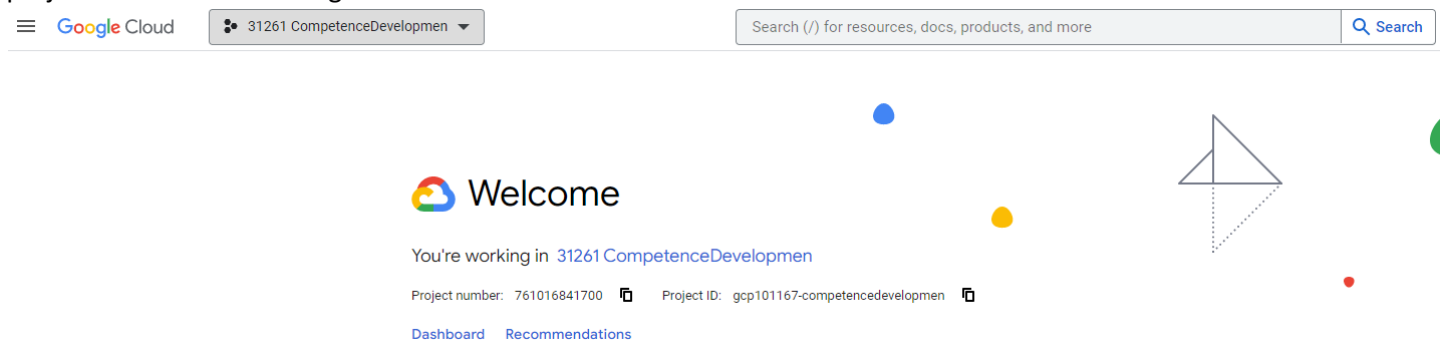


Fig. 2.2. Project created on Google Cloud

### 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.

Cloud Storage

Buckets

Monitoring

NEW

Settings

Buckets

+

CREATE

↺

REFRESH

Filter

Filter buckets

<input type="checkbox"/>	Name <div>↑</div>	Created	Location type	Location	Default storage class <div>?</div>
<input type="checkbox"/>	<div></div>	Mar 22, 2023, 2:59:27 PM	Region	europe-central2	Standard
<input type="checkbox"/>	<div></div>	May 3, 2023, 3:23:40 PM	Multi-region	us	Standard
<input type="checkbox"/>	<div>pyspark-test-tasks-bucket</div>	Apr 20, 2023, 11:10:41 AM	Region	europe-central2	Standard

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/](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.

pyspark-test-tasks-bucket					
Location	Storage class	Public access	Protection		
europe-central2 (Warsaw)	Standard	Not public	None		
OBJECTS	CONFIGURATION	PERMISSIONS	PROTECTION	LIFECYCLE	OBSERVABILITY
Buckets > pyspark-test-tasks-bucket					
UPLOAD FILES	UPLOAD FOLDER	CREATE FOLDER	TRANSFER DATA	MANAGE HOLDS	DOWNLOAD
Filter by name prefix only	Filter	Filter objects and folders			
<input type="checkbox"/>	Name	Size	Type	Created ?	Storage class
<input type="checkbox"/>	init.sh	1.9 KB	text/x-sh	May 9, 2023, 12:18:39 PM	Standard
<input type="checkbox"/>	py_spark_test_tasks/	–	Folder	–	–
<input type="checkbox"/>	pyspark_airflow.py	7.5 KB	text/x-python	May 9, 2023, 5:16:47 PM	Standard

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.

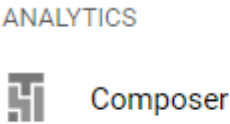


Fig. 4.1. Composer service ribbon

## 4.2. Access Cloud Composer

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

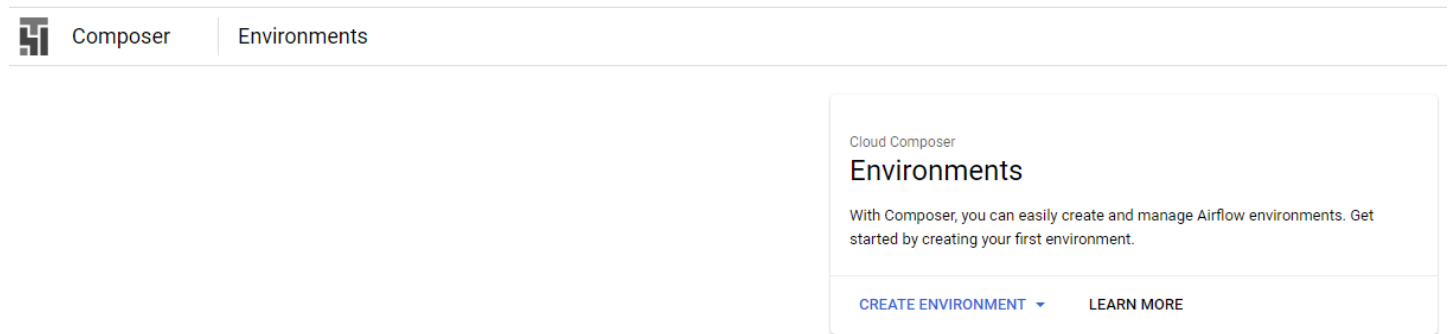


Fig. 4.2. Composer service environment

## 4.3. Create Cloud Composer

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

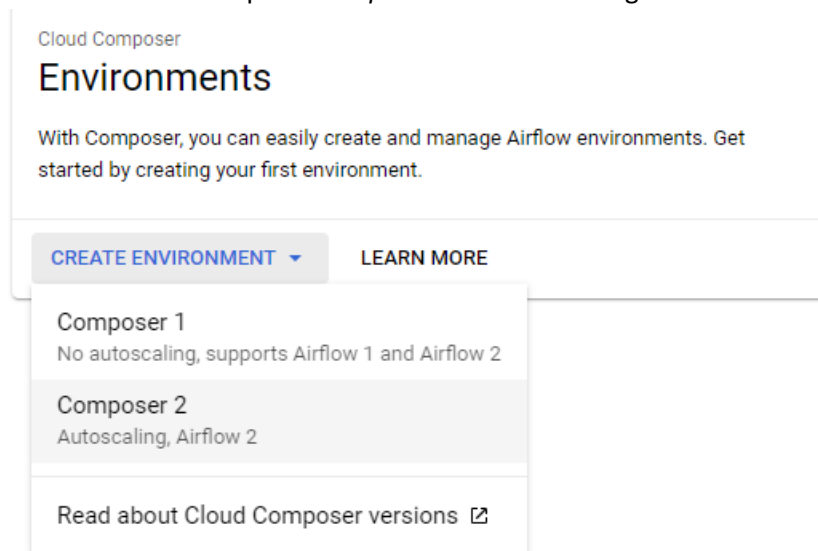


Fig. 4.3. Create Composer service environment

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.

State	Name ↑	Location	Composer version	Airflow version	Creation time	Update time	Airflow webserver	DAG list	Logs	DAGs folder
<input checked="" type="checkbox"/>	<a href="#">composer-pyspark-task</a>	europe-central2	1.20.12	1.10.15	5/16/23, 12:42 PM	5/16/23, 12:57 PM	<a href="#">Airflow</a>	<a href="#">DAGs</a>	<a href="#">Logs</a>	<a href="#">DAGs</a>

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.



**Name \***  
composer-pyspark-task

**Location \***  
europe-central2  

Google Cloud Platform region in which the environment will be created.

**Image version \***  
composer-1.20.12-airflow-1.10.15 

The image version to use in the created environment. Image version value includes Composer version and Airflow version. Must specify location before selecting image version.

## Node configuration

The configuration information for the Google Kubernetes Engine nodes running the Airflow software.

**Node count \***  
3

The number of nodes in the Google Kubernetes Engine cluster that will be used to run this environment.

**Zone**  
europe-central2-b 

Google Cloud Platform zone in which to deploy cluster nodes. Must specify location before selecting zone. [Learn more.](#)

**Machine type**  
n1-standard-2 

The Google Compute Engine machine type used for cluster instances. If unspecified, the machine type will default to 'n1-standard-1'. Must specify location before selecting machine type. [Learn more.](#)

**Disk size (GB)**  
30

The disk size in GB used for node VMs. Minimum is 30 GB. If unspecified, defaults to 100 GB. Cannot be updated.

**OAuth Scopes**

The set of Google API scopes to be made available on all node VMs. If empty, defaults to <https://www.googleapis.com/auth/cloud-platform>. Cannot be updated. [Learn more.](#)

**Service account**  
761016841700-compute@developer.gserviceaccount.com 

The Google Cloud Platform Service Account to be used by the node VMs. If a service account is not specified, the "default" Compute Engine service account is used. Cannot be updated.

**Tags**

The list of instance tags applied to all node VMs. Tags are used to identify valid sources or targets for network firewalls. Each tag with the list must comply with RFC1035. Cannot be updated.

**Number of schedulers \***  
1

Your environment can run more than one Airflow scheduler at the same time.

✓ NETWORKING, AIRFLOW CONFIG OVERRIDES, AND ADDITIONAL FEATURES.

## Beta API

Composer Beta API provides a set of functionality that is under preview and is not covered by any SLA and deprecation policy.

☐ Enforce the usage of Beta API **PREVIEW**

**CREATE**

CANCEL

Fig. 4.5. Setup Composer service environment

DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
airflow_monitoring	*/10 * * * *	airflow		2023-05-16 10:10		

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.

europa-central2-composer-py-5d01d011-bucket

Location	Storage class	Public access	Protection
europa-central2 (Warsaw)	Standard	Subject to object ACLs	None

OBJECTS CONFIGURATION PERMISSIONS PROTECTION LIFECYCLE OBSERVABILITY INVENTORY REPORTS NEW

Buckets > europa-central2-composer-py-5d01d011-bucket > dags

UPLOAD FILES UPLOAD FOLDER CREATE FOLDER TRANSFER DATA MANAGE HOLDS DOWNLOAD DELETE

Filter by name prefix only Filter Filter objects and folders

Name	Size	Type	Created	Storage class	Last modified	Public access
airflow_monitoring.py	809 B	text/x-python	May 16, 2023, 12:53:50 PM	Standard	May 16, 2023, 12:53:50 PM	Not public

Fig. 4.8. Initial Cloud Composer DAGs folder content

Name	Size	Type	Created
airflow_monitoring.py	809 B	text/x-python	May 16, 2023, 12:53:50 PM
pyspark_airflow.py	7.9 KB	text/x-python	May 16, 2023, 1:42:32 PM

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.

DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
airflow_monitoring	*/10 * * * *	airflow		2023-05-16 10:30		
pyspark_tasks_workflow	1 day, 0:00:00	airflow		2023-05-16 10:46		

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.

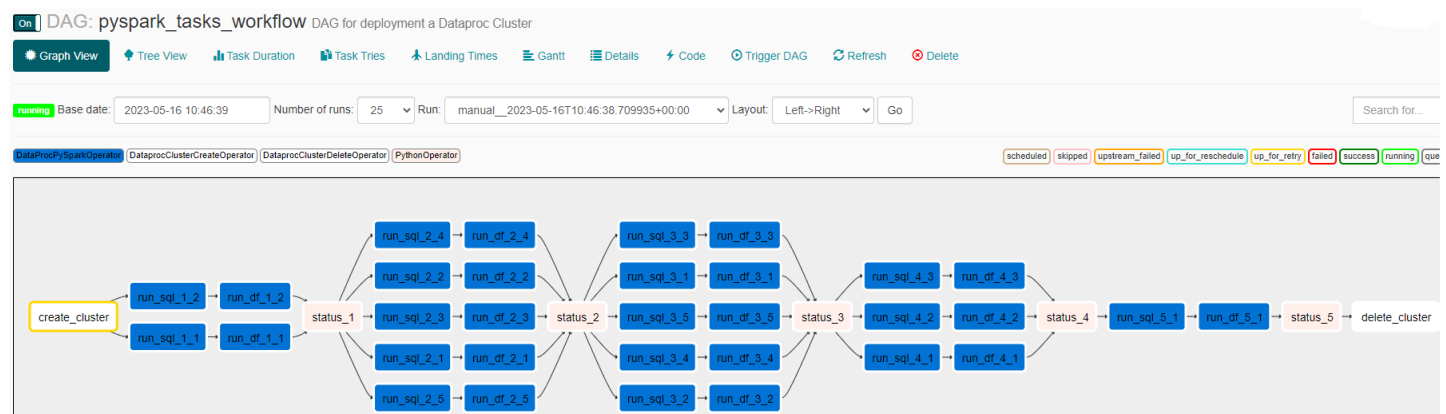


Fig. 4.11. Airflow specific DAG

## 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.

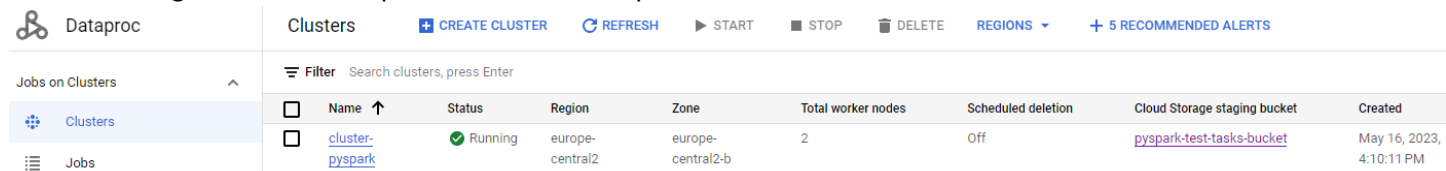


Fig. 5.1. Dataproc cluster

### 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.

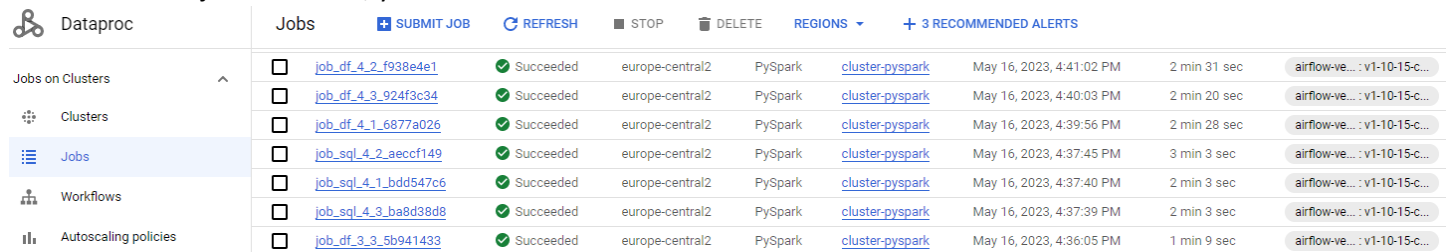


Fig. 5.2. Dataproc jobs

### 5.3. Monitor Cloud Storage logs

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

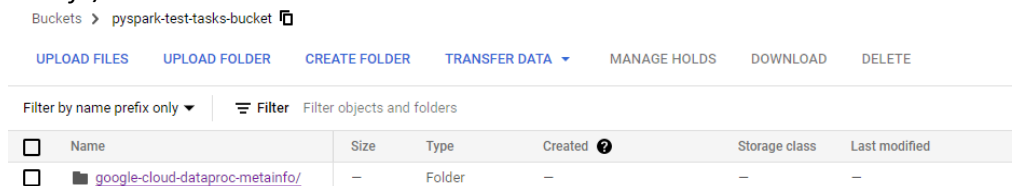


Fig. 5.3. Logs from Dataproc

### 5.4. Delete Cloud Composer

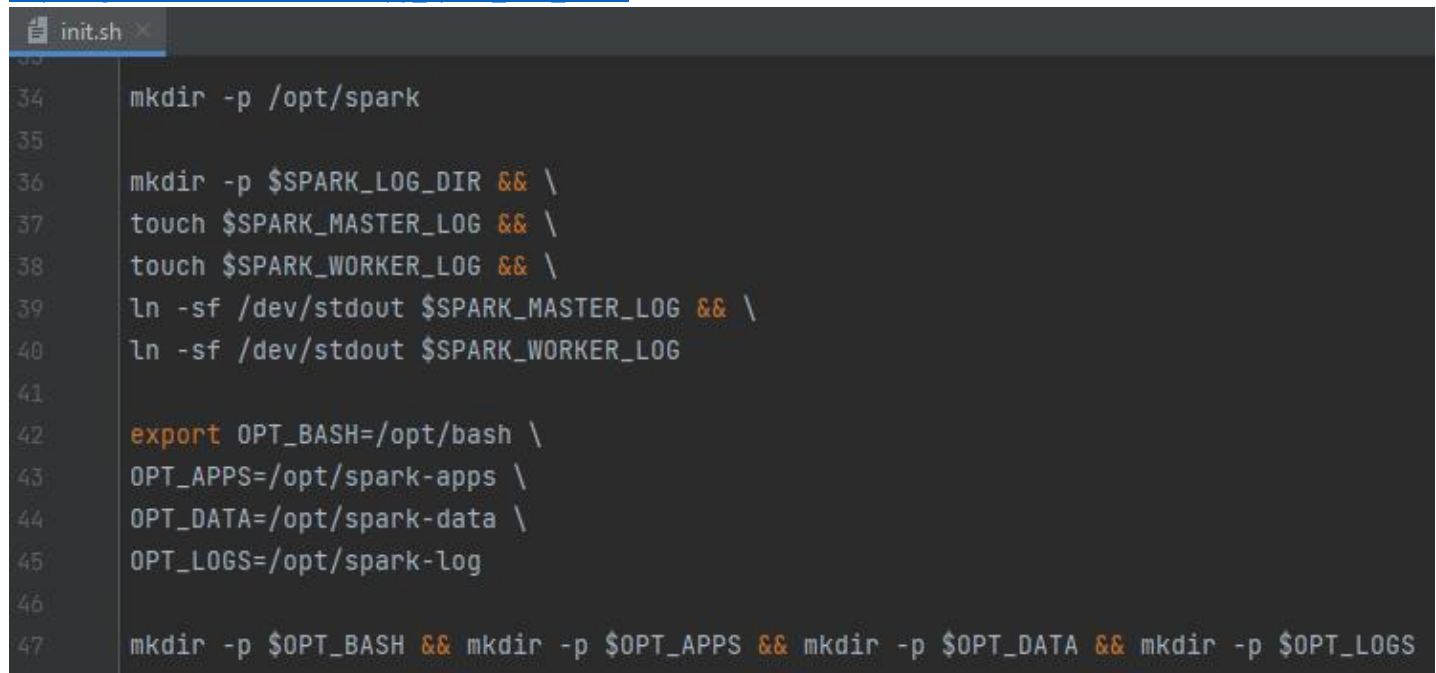
After all the jobs were performed it is recommended to delete the Cloud Composer by selecting the Composer and choose from top menu Delete option. The deletion process will take round 15 minutes.

## 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](https://github.com/maleterian/py_spark_test_tasks).

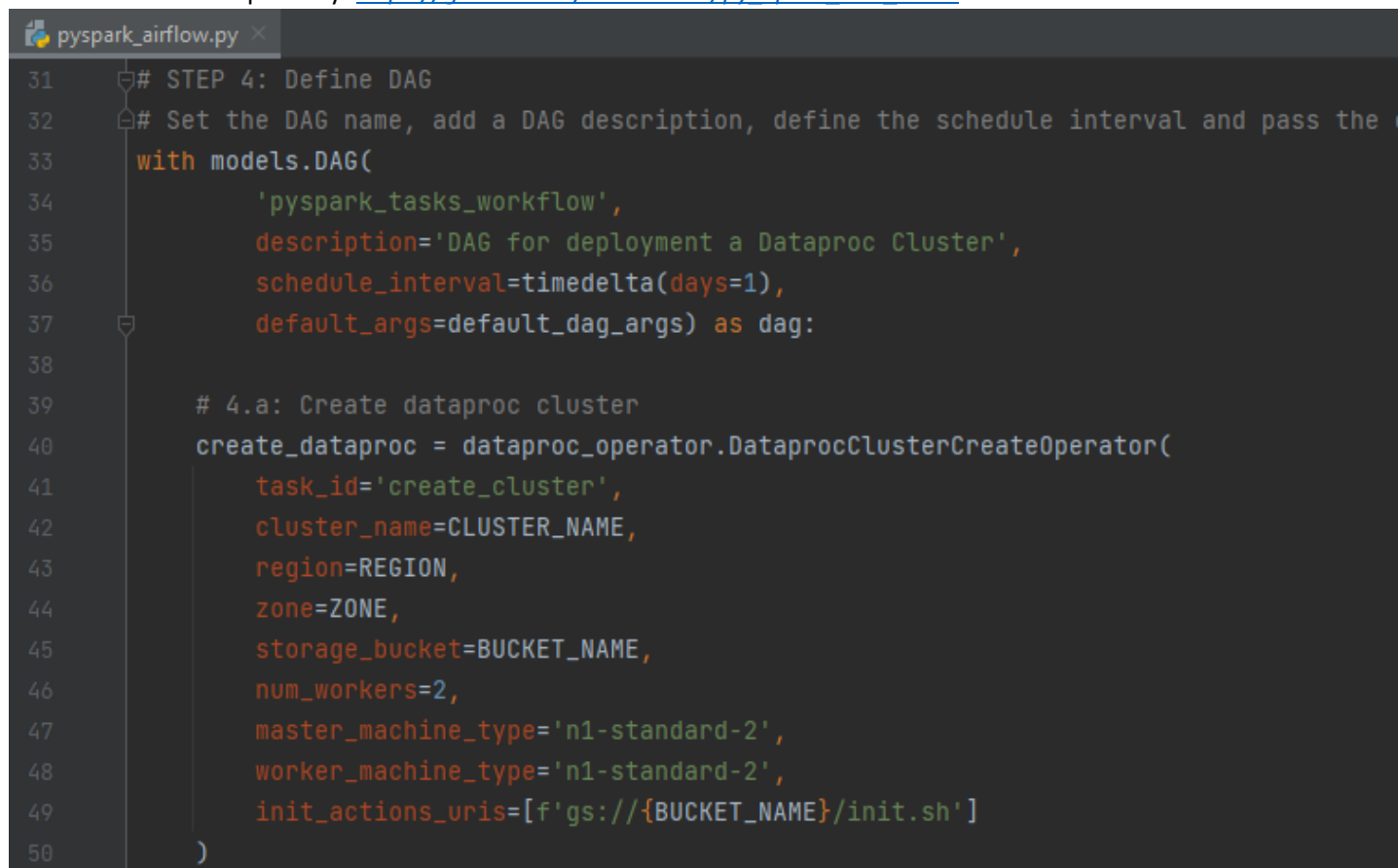


```
34 mkdir -p /opt/spark
35
36 mkdir -p $SPARK_LOG_DIR && \
37 touch $SPARK_MASTER_LOG && \
38 touch $SPARK_WORKER_LOG && \
39 ln -sf /dev/stdout $SPARK_MASTER_LOG && \
40 ln -sf /dev/stdout $SPARK_WORKER_LOG
41
42 export OPT_BASH=/opt/bash \
43 OPT_APPS=/opt/spark-apps \
44 OPT_DATA=/opt/spark-data \
45 OPT_LOGS=/opt/spark-log
46
47 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](https://github.com/maleterian/py_spark_test_tasks).



```
31 # STEP 4: Define DAG
32 # Set the DAG name, add a DAG description, define the schedule interval and pass the
33 with models.DAG(
34     'pyspark_tasks_workflow',
35     description='DAG for deployment a Dataproc Cluster',
36     schedule_interval=timedelta(days=1),
37     default_args=default_dag_args) as dag:
38
39     # 4.a: Create dataproc cluster
40     create_dataproc = dataproc_operator.DataprocClusterCreateOperator(
41         task_id='create_cluster',
42         cluster_name=CLUSTER_NAME,
43         region=REGION,
44         zone=ZONE,
45         storage_bucket=BUCKET_NAME,
46         num_workers=2,
47         master_machine_type='n1-standard-2',
48         worker_machine_type='n1-standard-2',
49         init_actions_uris=[f'gs://{BUCKET_NAME}/init.sh']
50     )
```

Fig. 6.2. Code snippet from pyspark\_airflow.py