2019-fall-group-project-ricardo-jacob-khalil

2019-fall-group-project-Ricardo-Jacob-Khalil created by GitHub Classroom

Project Members

- Jacob Jablonski
- Khalil Hanna
- Ricardo Rocha

Communication Platform

- [Slack (https://app.slack.com/client/THRKMGT55/GPHGQ9SH0)](https://app.slack.com/client/THRKMGT55/GPHGQ9SH0)

Contents

Collisions Routieres Dataset

Description

List of collisions that have occurred in Montreal since 2012.

This set includes collisions involving at least one motor vehicle circulating on the network and which have been the subject of a police report. It includes descriptive, contextual and event location information, including seriousness in terms of death, serious injury, minor injury and property damage only.

Dataset Source

[Collisions Routieres (http://donnees.ville.montreal.qc.ca/dataset/collisions-routieres)](http://donnees.ville.montreal.qc.ca/dataset/collisions-routieres)

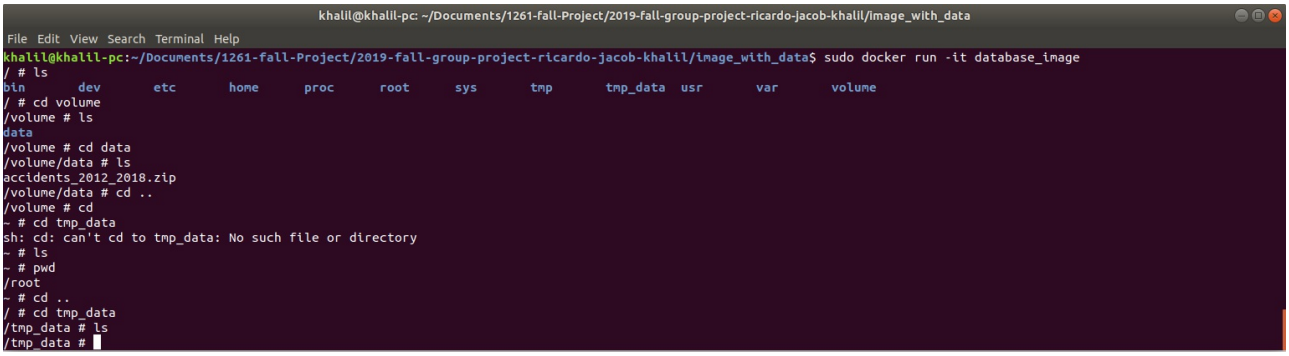
Dataset Characteristics

Number of Instances: 171,271
Number of Attributes: 68
Publishers: Service de l'urbanisme et de la mobilité - Direction de la mobilité
Frequency of update: Annual
Language: French
Geographic coverage: Territory of the city of Montreal
Temporal coverage: 2012-01-01 / 2018-12-31
Last edit: 2019-09-17 09:40
Created on: 2018-11-11 21:39
License: Creative Commons Attribution 4.0 International

Fetching the data and download it into volume using an image

using Dockerfile [data_dockerfile (image_with_data/data_dockerfile)](image_with_data/data_dockerfile) to build an image downloads the data from the source to tmp_data directory then move this data to data directory inside a volume when the container will be created by running the following commands:

```
docker build -t database_image -f data_dockerfile .
docker run -it database_image
```



Creating Volume

Create Data volume

```
docker volume create project-scripts-volume
```

Copy Database to Volume

```
docker run --rm -v project-scripts-volume:/volume database_image
```

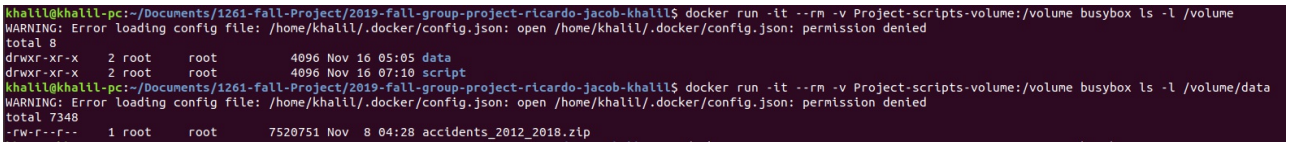
Copy Data Folder to Volume

```
docker run --rm -v "$(pwd)"/data:/data \
-v project-scripts-volume:/volume busybox \
cp -r /data/ /volume
```

Volume Contents

```
docker run -it --rm -v project-scripts-volume:/volume busybox ls -l /volume
```

```
docker run -it --rm -v project-scripts-volume:/volume busybox ls -l
/volume/data
```



Services on Spark Cluster

Create Spark Network

Using the following command a spark network will be created as "spark-network"

```
docker network create spark-network
```

Spark Cluster with HDFS and MongoDB

using docker compose to create spark cluster by running [spark-compose.yml (./spark-compose.yml)](./spark-compose.yml) file using the below command:

```
env user_mongo=root pass_mongo=password docker-compose --file spark-compose.yml
up --scale spark-worker=2
```

Preparing Data and Work Environment

Ckecking if the volume accessable by the cluster

- check the containers of the cluster
- execute the worker container to check the volume using the command:

```
docker exec -it containerID sh
```

```
khalil@khalil-pc: ~/Documents/1261-fall-Project/2019-fall-group-project-ricardo-jacob-khalil
File  Edit  View  Search  Terminal  Help
khalil@khalil-pc:~/Documents/1261-fall-Project/2019-fall-group-project-ricardo-jacob-khalil$ docker ps
WARNING: Error loading config file: /home/khalil/.docker/config.json: open /home/khalil/.docker/config.json: permission denied
CONTAINER ID        IMAGE                   COMMAND                  CREATED             STATUS              PORTS

                                                        NAMES
09b5e24f572c        mjhea0/spark:2.4.1      "bin/spark-class org…"    10 minutes ago      Up 10 minutes       0.0.0.0:8081->8081/tcp

                                                        2019-fall-group-project-ricardo-jacob-khalil_worker_1
085b8779c66d        harisekhon/hadoop:2.8   "/bin/sh -c \"/entryp…"   10 minutes ago      Up 10 minutes       0.0.0.0:8042->8042/tcp, 8020/tcp, 9
000/tcp, 0.0.0.0:8088->8088/tcp, 0.0.0.0:19888->19888/tcp, 0.0.0.0:50010->50010/tcp, 0.0.0.0:50020->50020/tcp, 0.0.0.0:50070->50070/tcp, 0.0.0.0:
50075->50075/tcp, 10020/tcp, 0.0.0.0:50090->50090/tcp   2019-fall-group-project-ricardo-jacob-khalil_hadoop_1
362bc220e884        mjhea0/spark:2.4.1      "bin/spark-class org…"    10 minutes ago      Up 10 minutes       0.0.0.0:4040->4040/tcp, 0.0.0.0:606
6->6066/tcp, 0.0.0.0:7077->7077/tcp, 0.0.0.0:8080->8080/tcp
                                                        2019-fall-group-project-ricardo-jacob-khalil_master_1
8ae1c5d69ec7        mongo-express           "tini -- /docker-ent…"    10 minutes ago      Up 10 minutes       0.0.0.0:8181->8081/tcp

                                                        2019-fall-group-project-ricardo-jacob-khalil_mongo-express_1
4aef60b50653        mongo                   "docker-entrypoint.s…"    10 minutes ago      Up 10 minutes       27017/tcp

                                                        2019-fall-group-project-ricardo-jacob-khalil_mongo_1
khalil@khalil-pc:~/Documents/1261-fall-Project/2019-fall-group-project-ricardo-jacob-khalil$ docker exec -it 09b5e24f572c sh
WARNING: Error loading config file: /home/khalil/.docker/config.json: open /home/khalil/.docker/config.json: permission denied
# ls /volume
data  script
# ls /volume/data
accidents_2012_2018.zip
#
```

Starting Spark Shell

Spark shell will be used to unzip the data inside the volume, upload the data on HDFS and MongoDB.
to start spark shell the below command will be run:

```
docker run -it --rm \
  -v project-scripts-volume:/volume \
  --network=spark-network \
  mjhea0/spark:2.4.1 \
  ./bin/pyspark \
  --master spark://master:7077 \
  --packages org.mongodb.spark:mongo-spark-connector_2.11:2.4.0
```

```
khalil@khalil-pc: ~
File  Edit  View  Search  Terminal  Help
        ---------------------------------------------------------------------
        |                  |            modules            ||   artifacts   |
        |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
        ---------------------------------------------------------------------
        |      default     |   2   |   2   |   2   |   0   ||   2   |   2   |
        ---------------------------------------------------------------------
:: retrieving :: org.apache.spark#spark-submit-parent-de76b109-c170-4f42-94b3-af7322acea7a
        confs: [default]
        2 artifacts copied, 0 already retrieved (2833kB/15ms)
19/11/18 19:52:38 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... usi
ng builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.4.1
      /_/

Using Python version 3.4.2 (default, Feb  7 2019 06:08:06)
SparkSession available as 'spark'.
>>>
```

Unzip the Data File:

using the following command in spark shell:

```
from zipfile import *
with ZipFile("/volume/data/accidents_2012_2018.zip", 'r') as zipObj:
...     zipObj.extractall('/volume/data')
```

```
>>> from zipfile import *
>>> with ZipFile("/volume/data/accidents_2012_2018.zip", 'r') as zipObj:
...         zipObj.extractall('/volume/data')
...
>>>
```

Store the data in HDFS and MongoDB

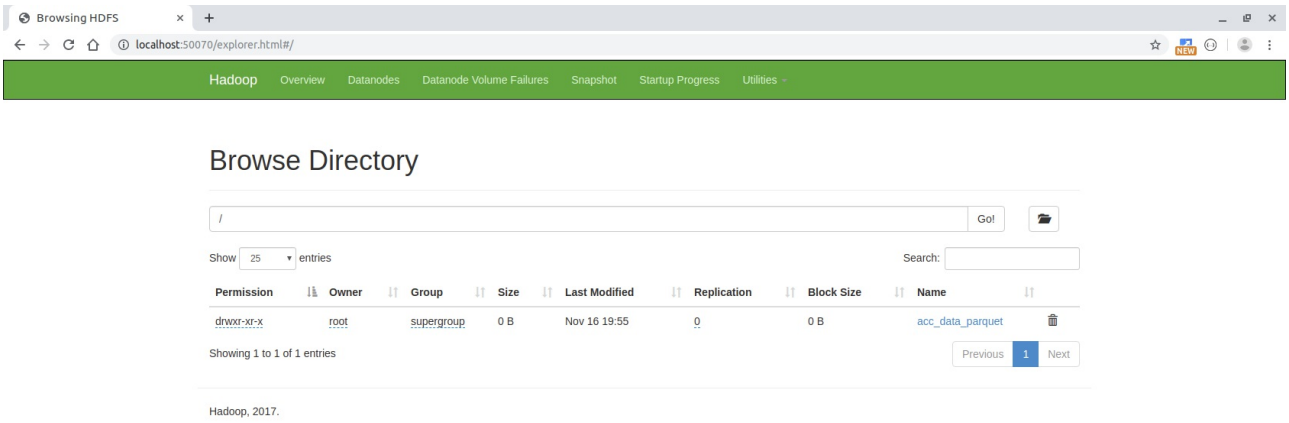
Store the data in HDFS as parquet

first read the unziped data from the volume then push it as parquet into HDFS. to acheive that the following command to be run in spark shell

```
acc_data = spark.read.csv("/volume/data")
acc_data.write.parquet("hdfs://hadoop/acc_data_parquet")
```

```
>>> acc_data = spark.read.csv("/volume/data/")
>>> acc_data.write.parquet("hdfs://hadoop/acc_data_parquet")
19/11/17 00:54:54 WARN util.Utils: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by sett
ing 'spark.debug.maxToStringFields' in SparkEnv.conf.
>>>
```

to check the data file on HDFS open [http://localhost:50070 (http://localhost:50070)](http://localhost:50070) then navigate to "Utilities" in main bar, select "Browse the file system" then the below page will open.



Store the data in Mongodb

first in spark shell run the folowing comand to be able pushing the data to MongoDB

```
spark = SparkSession \
        .builder \
        .appName("mongodb") \
        .master("spark://master:7077") \
        .config("spark.mongodb.input.uri",
"mongodb://root:password@mongo/test.coll?authSource=admin") \
        .config("spark.mongodb.output.uri",
"mongodb://root:password@mongo/test.coll?authSource=admin") \
        .config('spark.jars.packages', 'org.mongodb.spark:mongo-spark-
connector_2.11:2.4.0')\
        .getOrCreate()
```

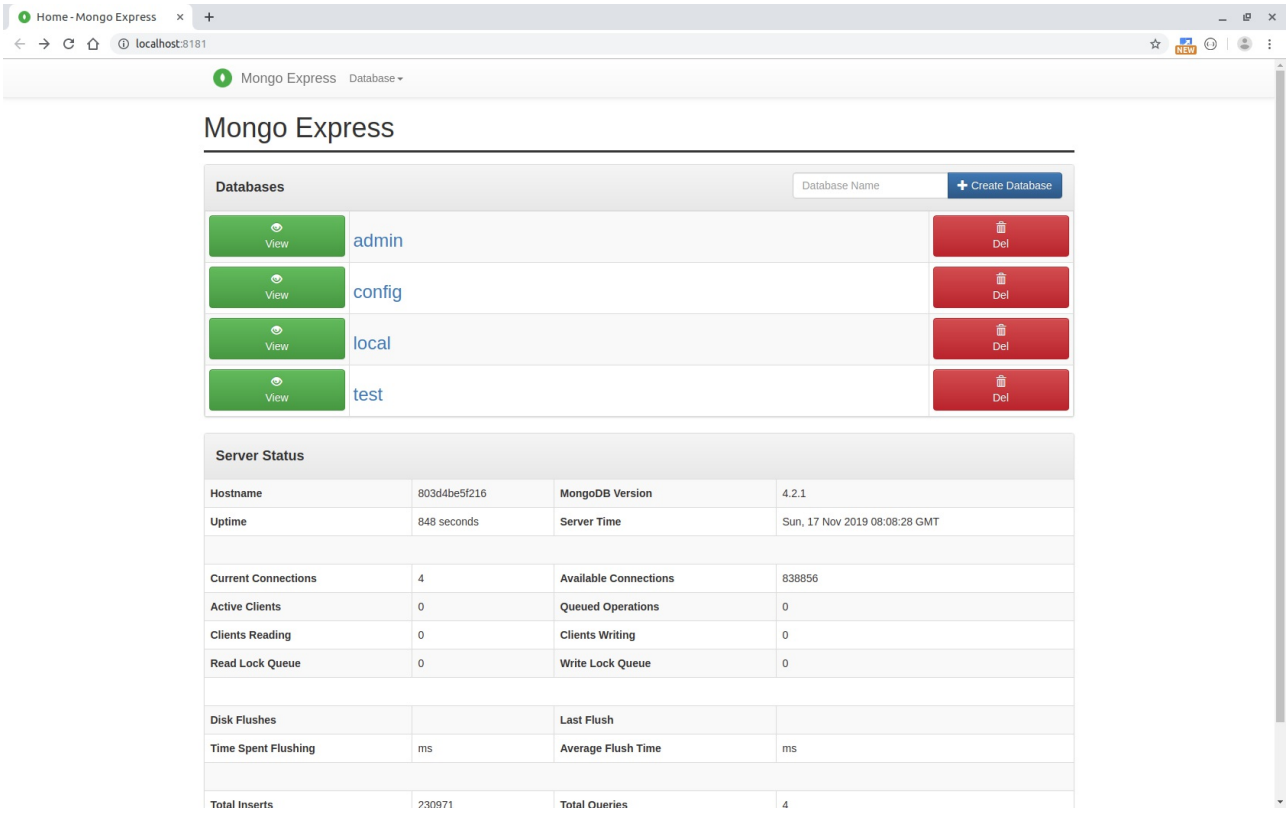
Read the data from the volume and store it in MongoDB

```
acc_mongo = spark.read.csv("/volume/data")
acc_mongo.write.format("com.mongodb.spark.sql.DefaultSource").mode("append").sa
ve()
```

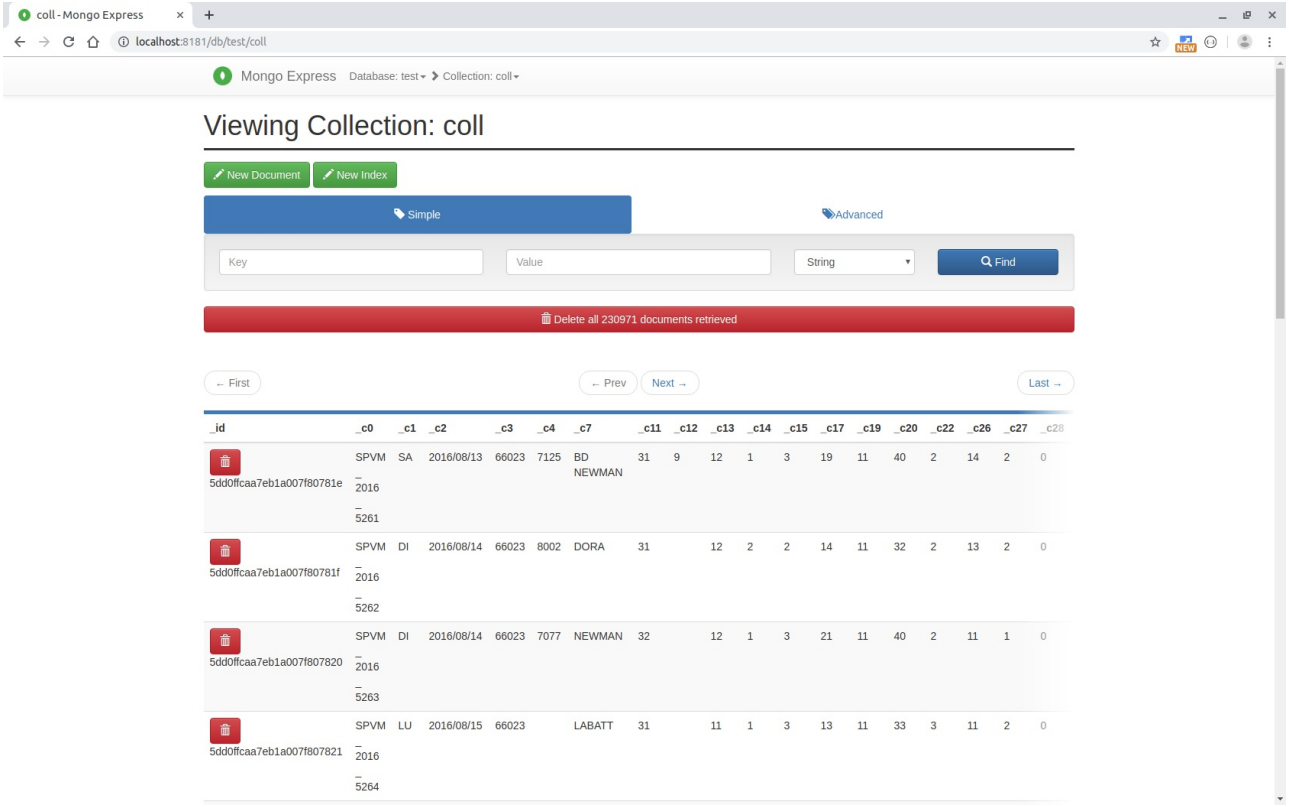
Read the data from the HDFS and store it in MongoDB

```
acc_mongo = spark.read.csv("hdfs://hadoop/acc_data_parquet")
acc_mongo.write.format("com.mongodb.spark.sql.DefaultSource").mode("append").sa
ve()
```

Open [http://localhost:8181 (http://localhost:8181)](http://localhost:8181) for Mongo Express.



Then click on "test" for database



Automated Storage

Copy the scripts into the volume

In the volume a directory named "script" will be created and copied all the required scripts into that directory by executing the following command:

```
docker run --rm -v "$(pwd)"/scripts:/script \
-v project-scripts-volume:/volume busybox \
cp -r /script/ /volume
```

Checking the scripts in the volume:

```
docker run -it --rm -v project-scripts-volume:/volume busybox ls -l
/volume/script
```

```
khalil@khalil-pc:~/Documents/1261-fall-Project/2019-fall-group-project-ricardo-jacob-khalil$ sudo docker run -it --rm -v project-scripts-volume:/volume busybox ls -l
/volume/script
total 8
-rw-r--r--    1 root     root           924 Nov 19 05:30 hdfs_store.py
-rw-r--r--    1 root     root          1189 Nov 19 05:30 mongodb_store.py
khalil@khalil-pc:~/Documents/1261-fall-Project/2019-fall-group-project-ricardo-jacob-khalil$
```

Store the data in HDFS as parquet

By execute [hdfs_store.py (./scripts/hdfs_store.py)](./scripts/hdfs_store.py) script as following:

```
docker run -t --rm \
  -v project-scripts-volume:/volume \
  --network=spark-network \
  mjhea0/spark:2.4.1 \
  bin/spark-submit \
    --master spark://master:7077 \
    --class endpoint \
    /volume/script/hdfs_store.py
```

Store the data in MongoDB

By execute [mongodb_store.py (./scripts/mongodb_store.py)](./scripts/mongodb_store.py) script as following:

```
docker run -t --rm \
  -v project-scripts-volume:/volume \
  --network=spark-network \
  mjhea0/spark:2.4.1 \
  bin/spark-submit \
    --master spark://master:7077 \
    --class endpoint \
    --packages org.mongodb.spark:mongo-spark-connector_2.11:2.4.0 \
    /volume/script/mongodb_store.py
```

Using Jupyter Notebook

Create Volume

Creating a volume to store the notbooks Which will be created

```
docker volume create notebooks
```

Jupyter in the Cluster

Keeping the cluster up now to deploy jupyter in cluster using docker compose by running [jupyter-compose.yml (jupyter-compose.yml)](jupyter-compose.yml) file using the below command::

```
env TOKEN=project1261 docker-compose --file jupyter-compose.yml up
```

Open http://localhost:8889/?token=project1261

jupyter have access to the volume where the data and scripts are stored.

Openning new notebook and prepare the environment.

Reading the Data from volume

Read the data from the volume as csv

Read Data from Volume

```
In [3]: acc_data = spark.read.csv("./volume/data")
```

```
In [11]: acc_data.printSchema()
```

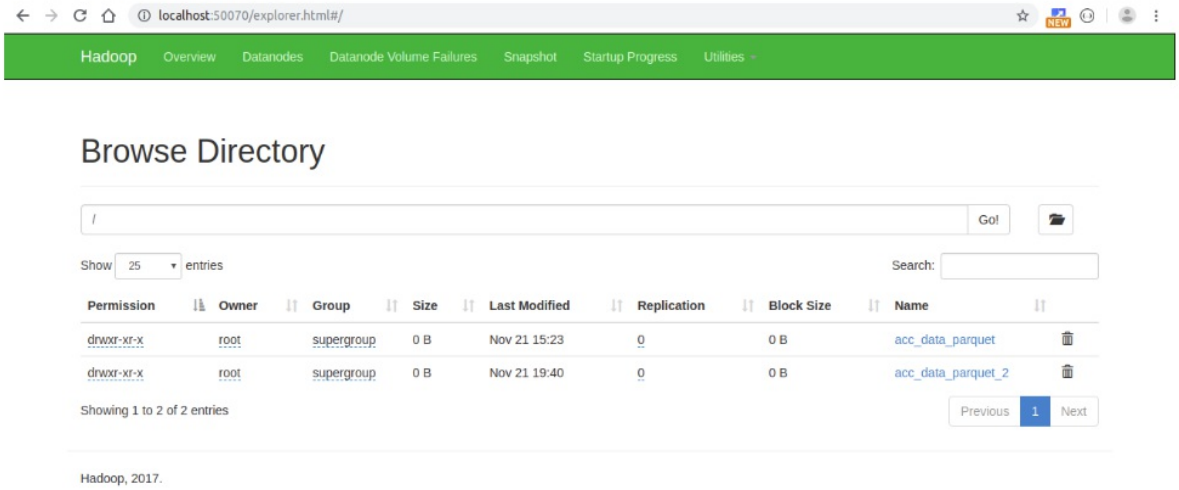
```
root
 |-- _c0: string (nullable = true)
 |-- _c1: string (nullable = true)
 |-- _c2: string (nullable = true)
 |-- _c3: string (nullable = true)
 |-- _c4: string (nullable = true)
 |-- _c5: string (nullable = true)
```

Store the Data on HDFS

Push the data to HDFS

Write Data on HDFS

```
In [4]: acc_data.write.parquet("hdfs://hadoop/acc_data_parquet_2")
```



Store the Data in MongoDB

Push the data from hdfs into MongoDB

Read The Data from HDFS

```
In [7]: acc_mongo_jupyter = spark.read.parquet("hdfs://hadoop/acc_data_parquet_2")
```

Push The Data to MongoDB

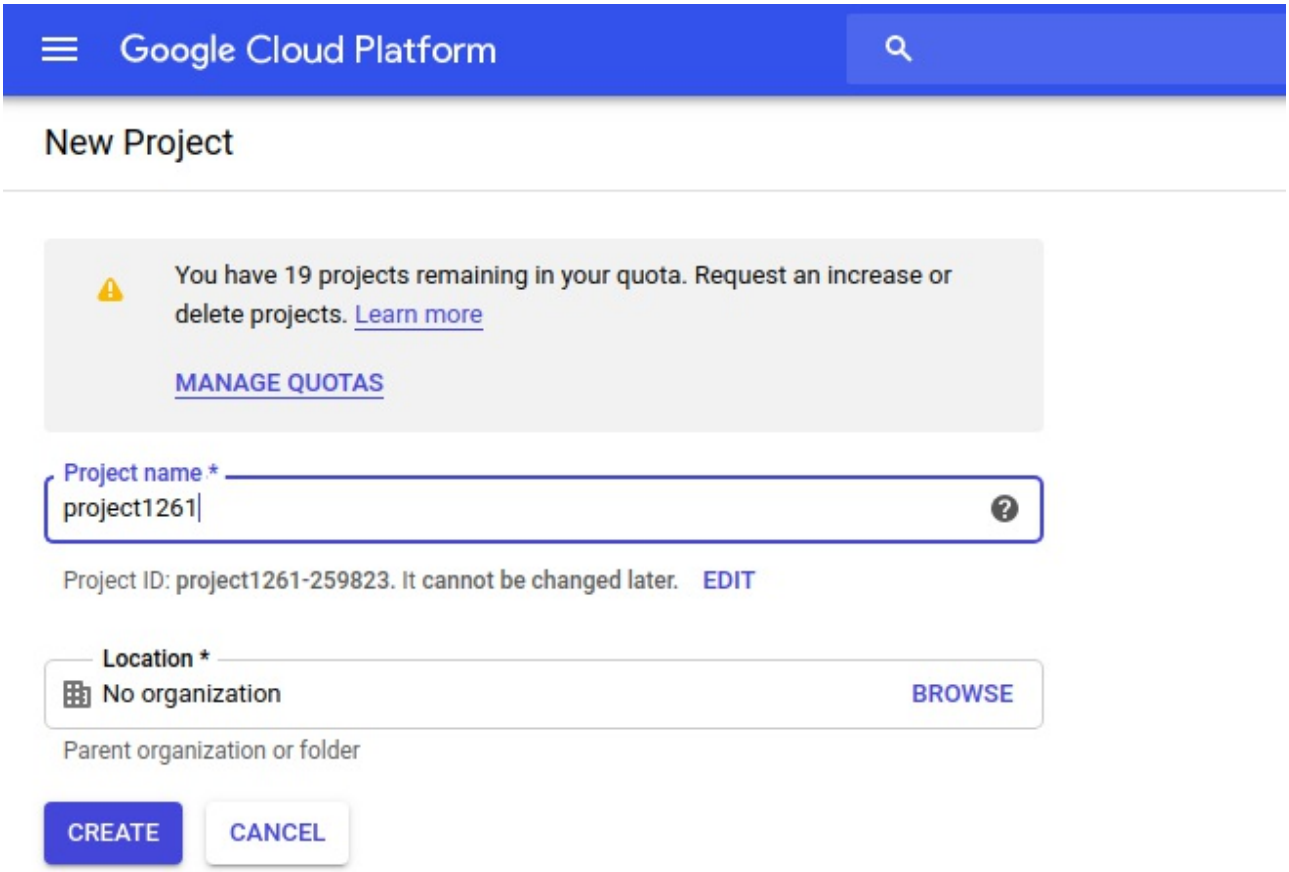
```
In [8]: acc_mongo_jupyter.write.format("com.mongodb.spark.sql.DefaultSource").mode("append").save()
```

[The Full Notebook (./hdfs_mongo_notebook.ipynb)](./hdfs_mongo_notebook.ipynb)

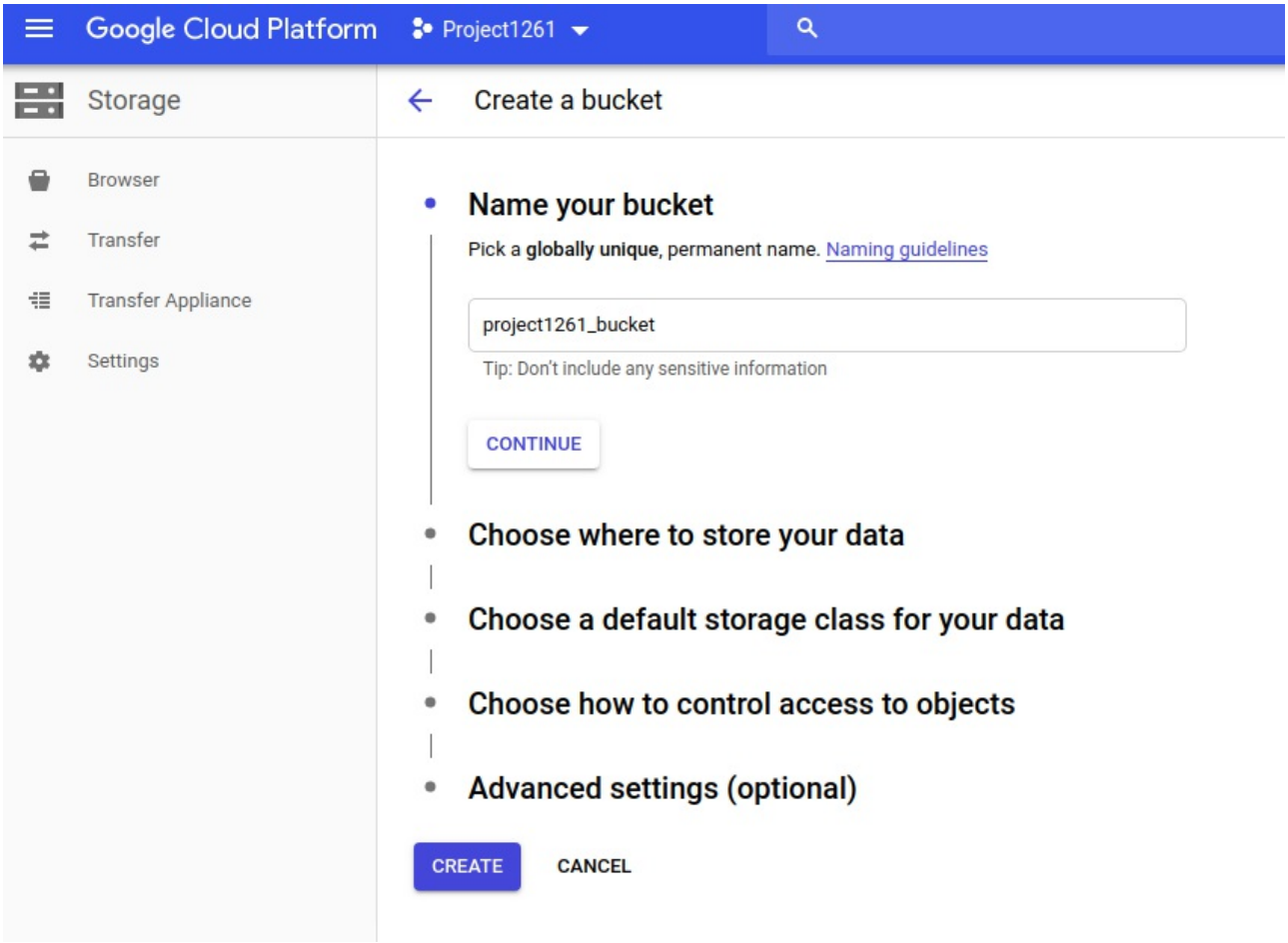
Google Cloud Platform

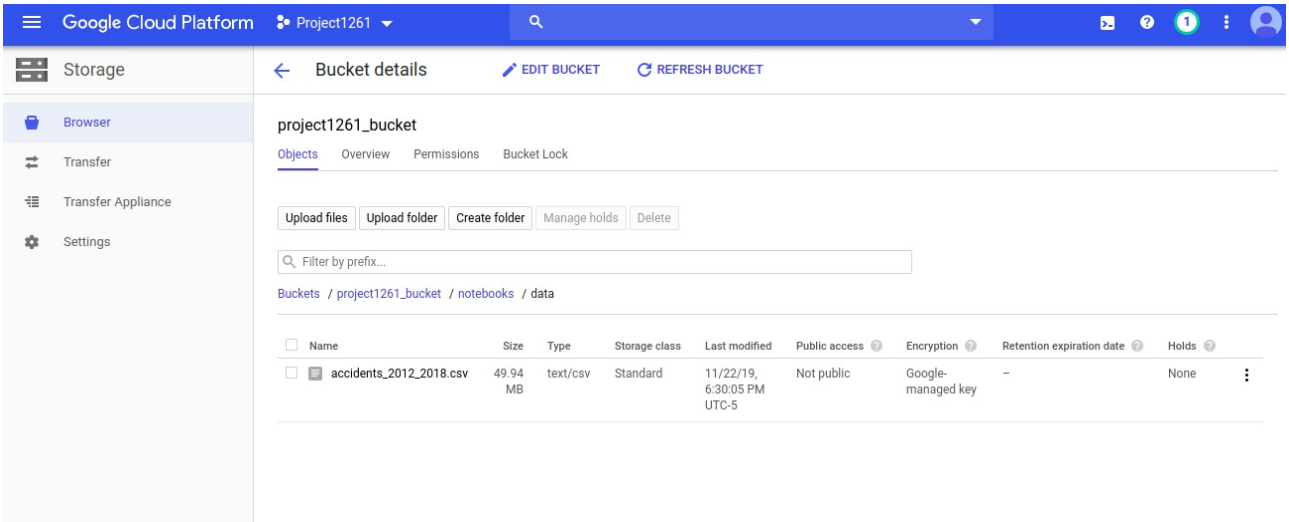
Environment Setup

- First lets create a project.



- Enabling the Cloud Dataproc and Google Compute Engine APIs.
- Creating bucket where notebook and the data will be stored.





- Creating cluster and selecting the the bucket to which was created to have access to the data and where the notebooks will be stored.

Google Cloud Platform Project1261

Dataproc

Clusters

Jobs

Workflows

Create a cluster

Name

project1261-cluster

Region

global

Zone

us-central1-c

Cluster mode

Standard (1 master, N workers)

Master node

Contains the YARN Resource Manager, HDFS NameNode, and all job drivers

Machine type

4 vCPUs 15 GB memory Customize

Primary disk size (minimum 15 GB)

500 GB

Primary disk type

Standard persistent disk

Worker nodes

Each contains a YARN NodeManager and a HDFS DataNode.
The HDFS replication factor is 2.

Machine type

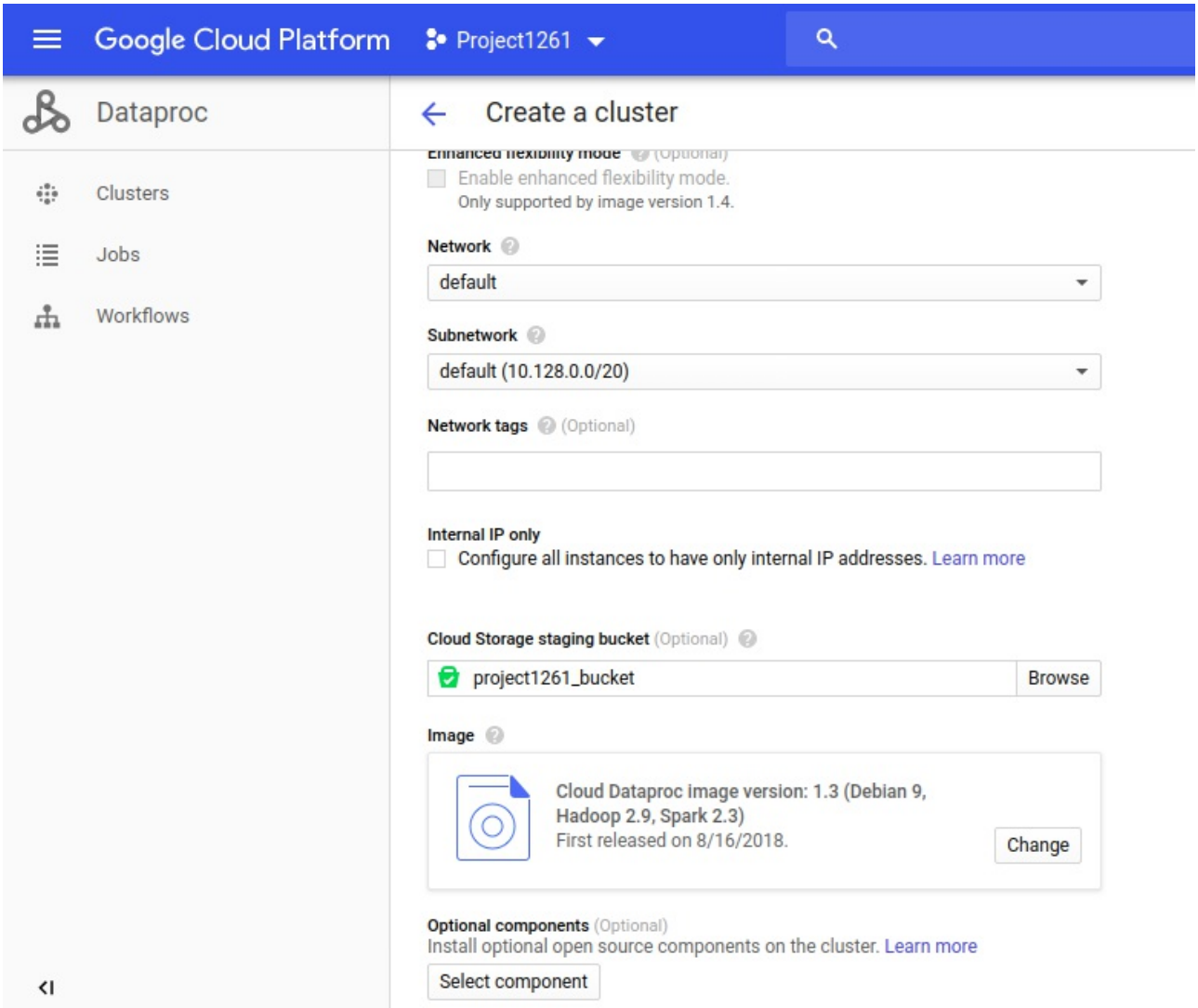
4 vCPUs 15 GB memory Customize

Primary disk size (minimum 15 GB)

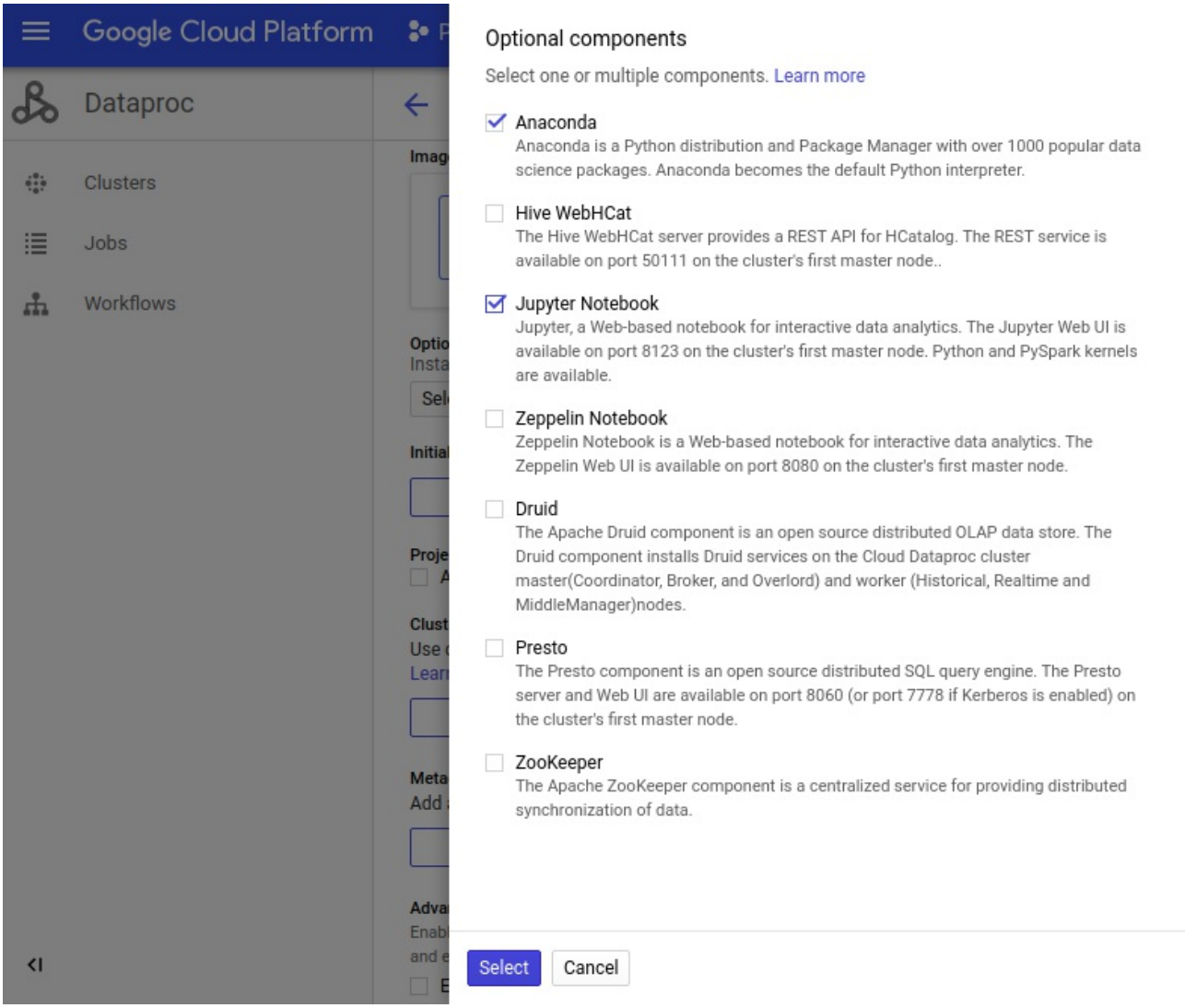
500 GB

Primary disk type

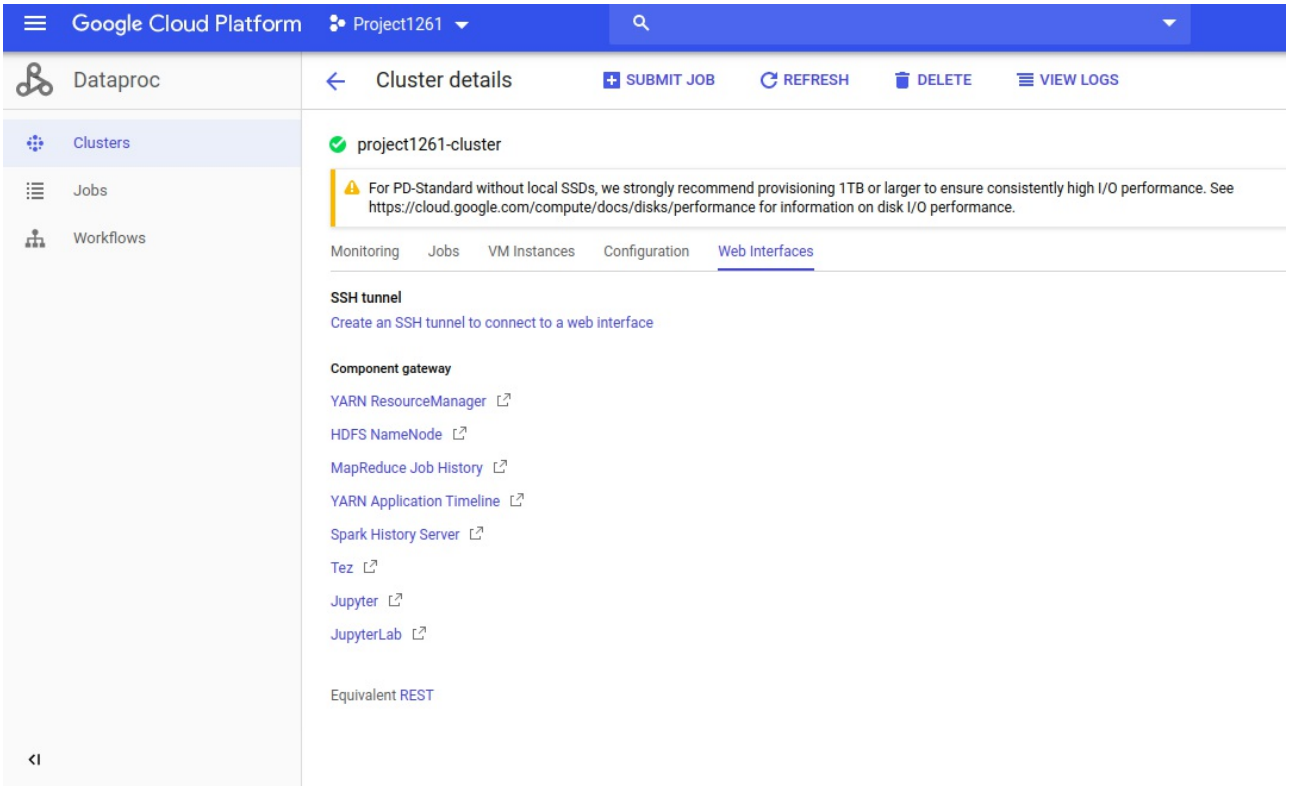
Standard persistent disk



In advanced options section Anaconda and jupyter components should be selected to run jupyter in the cluster.

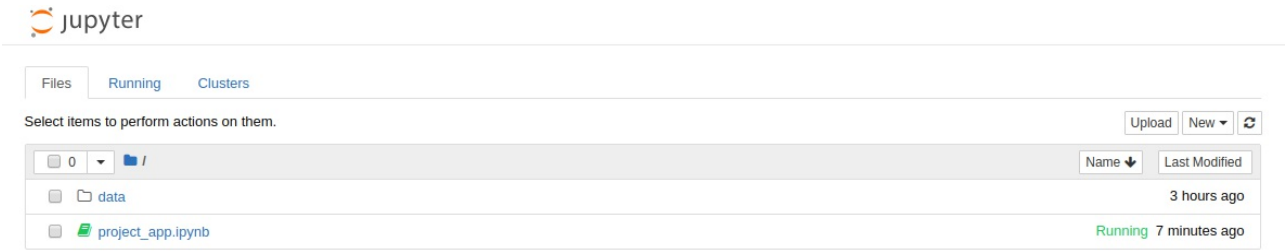


- From the web interfaces in created cluster we can open jupyter notebook which is runing on the cluster and have access to the bucket where our data is.



Analytics on GCP

Openning Jupyter on GCP cluster

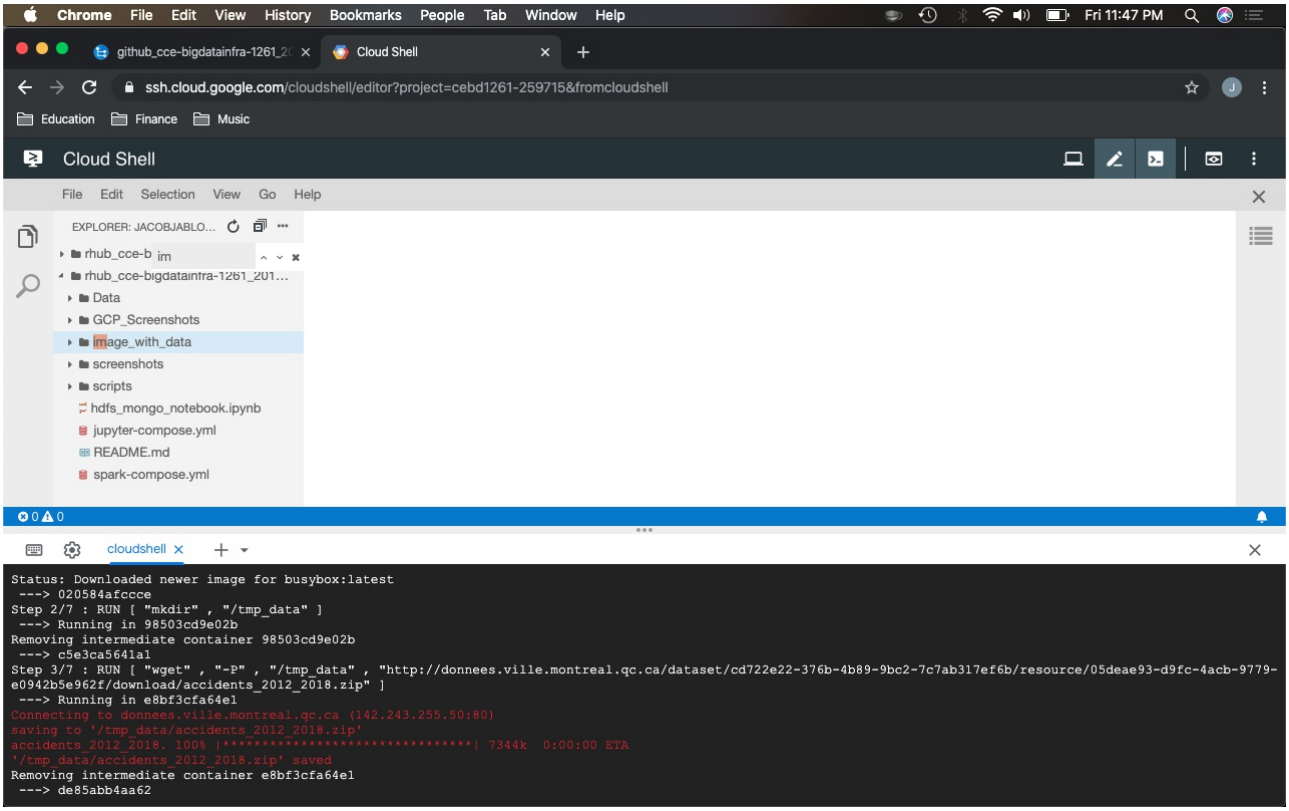


Running the Project App
[Exploratory Data Analysis on the Collisions Routieres dataset (./project_app.ipynb)]

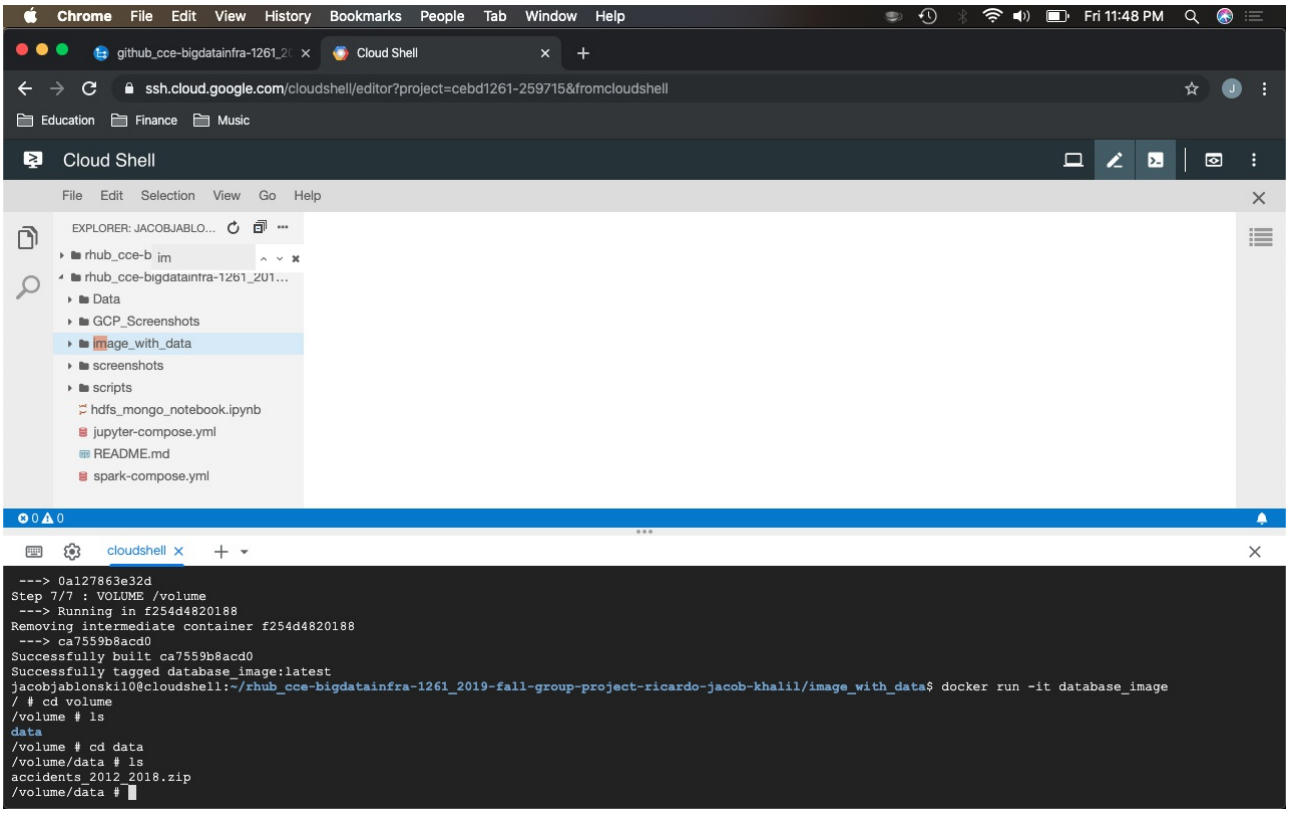
Github Mirror and Deploy via GCP

Using the exact same process as our local docker stack, we will deploy our application on the GCP. The only difference is that we skip the manual hdfs/mongo-express builds and go right to automated builds.

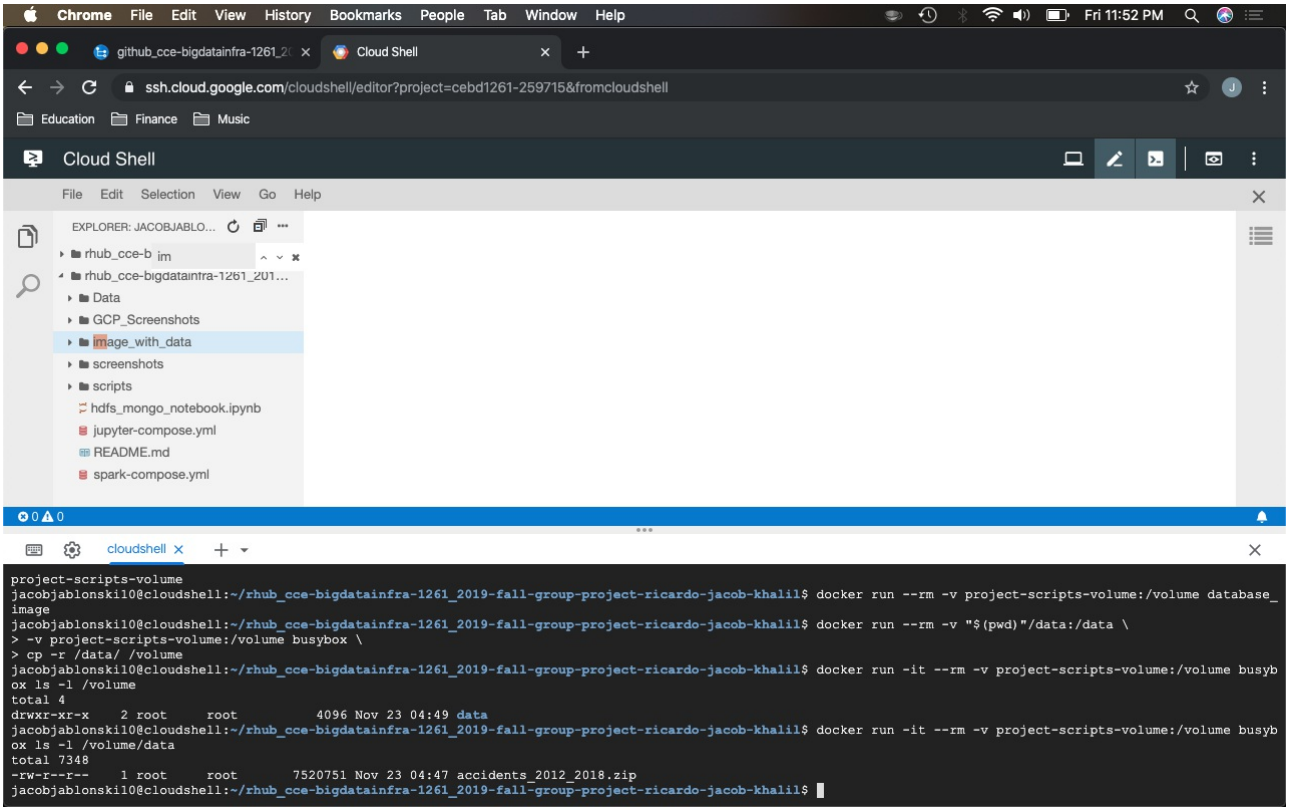
- Pulling data from github hosted docker image through google shell



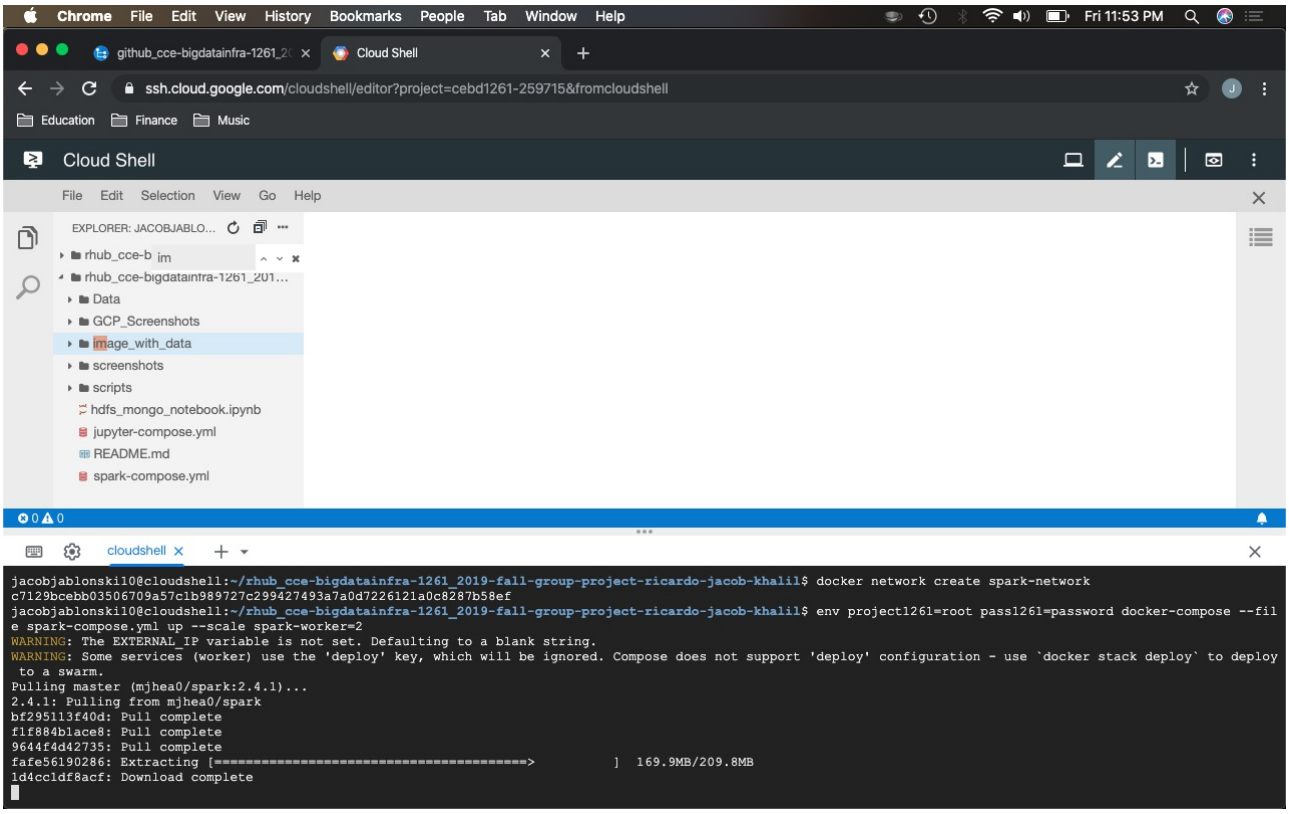
- Running the docker image and making sure the data is in the correct directory



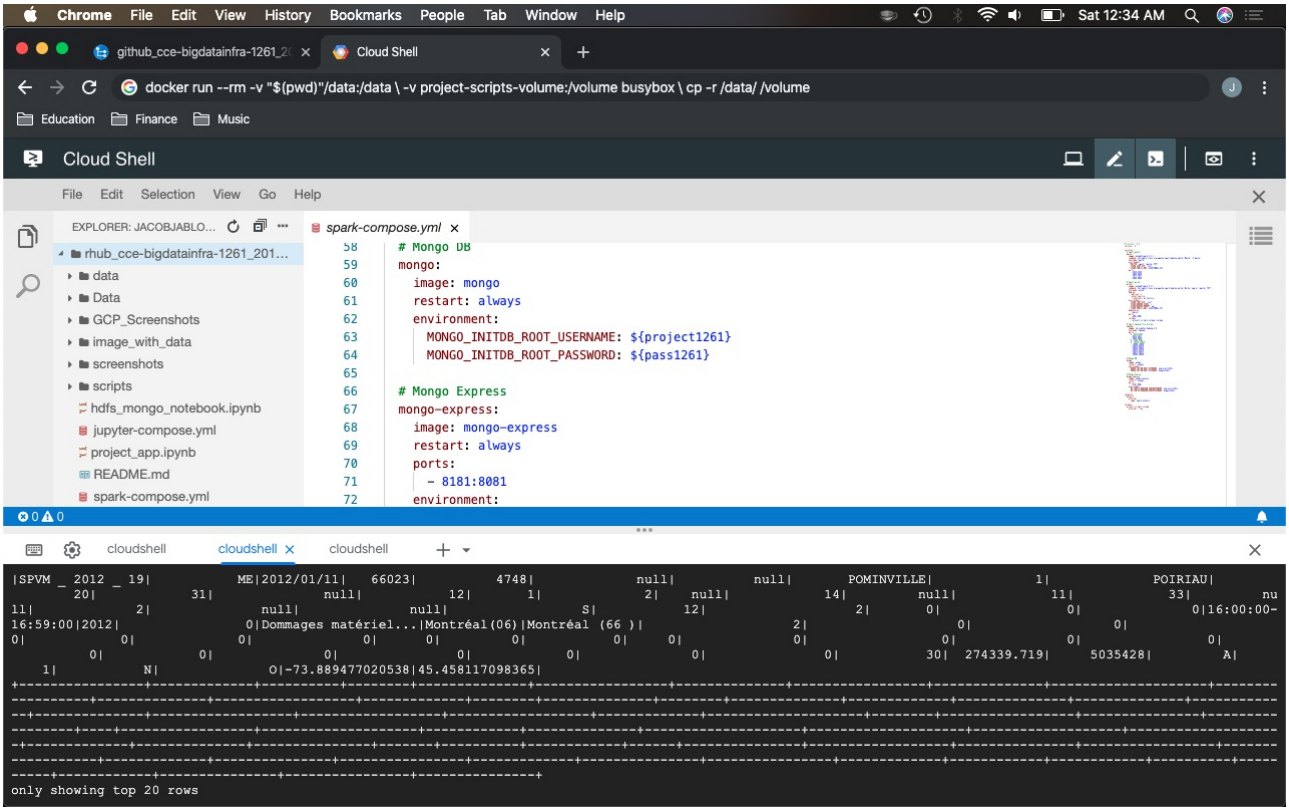
- Copying the scripts



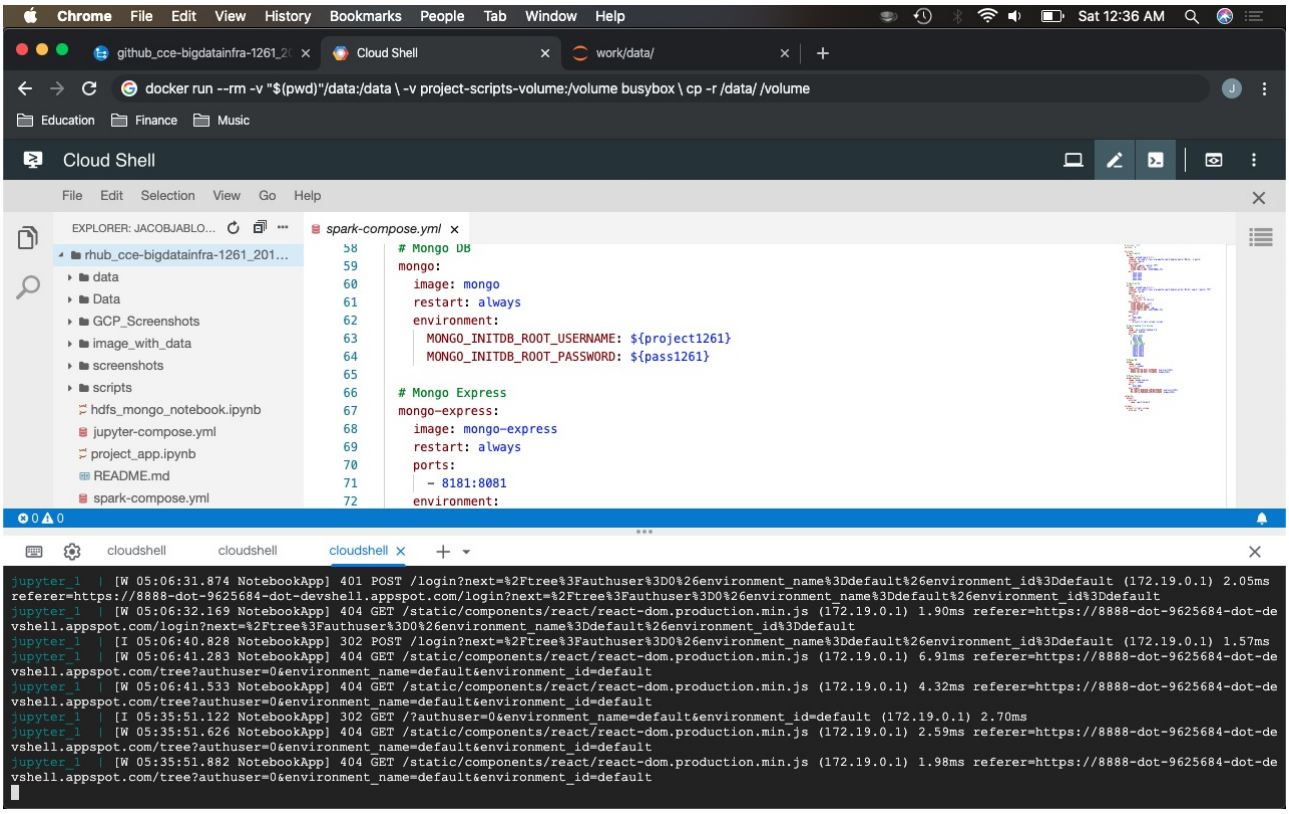
- Deploying the spark compose file with created network



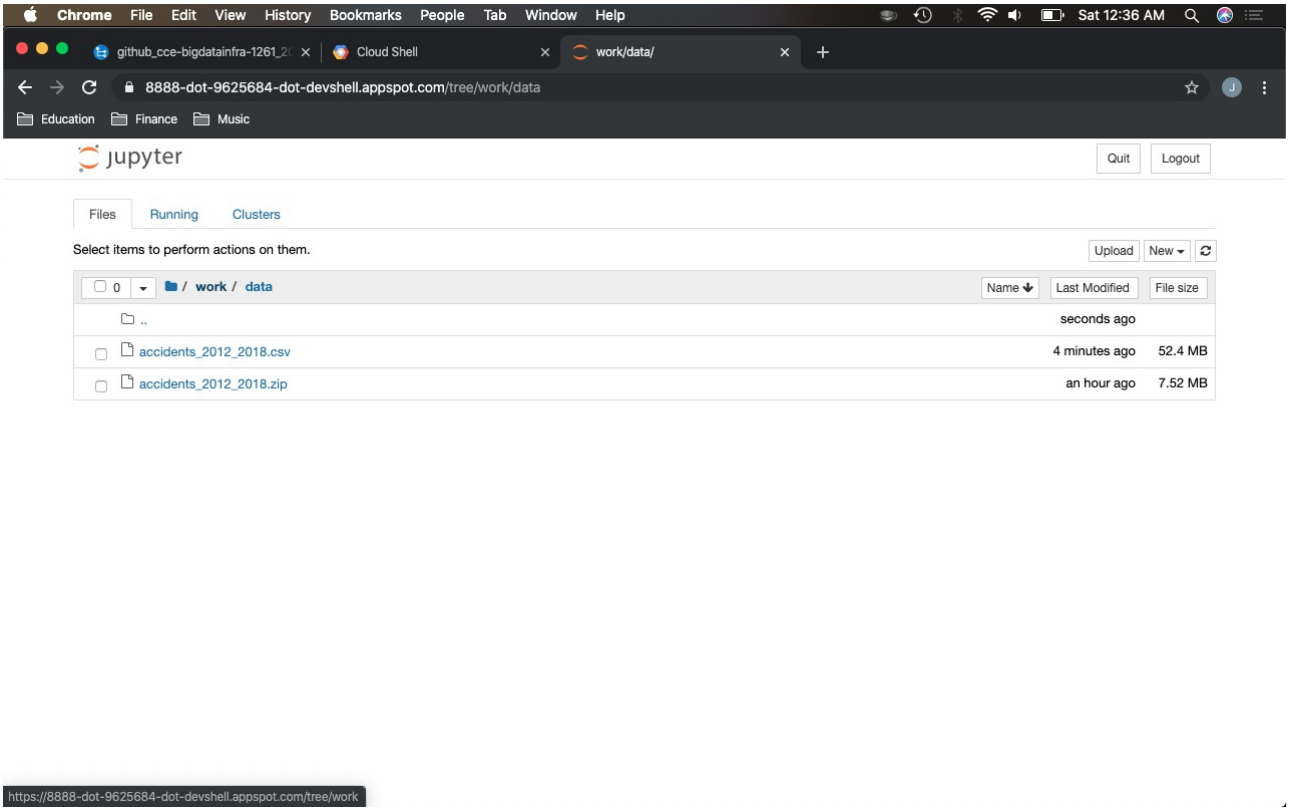
- Creating jupyter volume



- Jupyter compose up to be able to reach our notebook



- Proof of the notebook in action on our GCP project cluster



The Results Obtained

In our project, we chose to work with the dataset about traffic accidents that happened from 2012 to 2018 in the city of Montreal. We built our infrastructure using a docker container to create an image gathering that dataset from the Montreal Open Data Website. We decompressed, read and wrote the dataset in HDFS (Parquet files) and MongoDB using a spark console running on a Spark Cluster and making use of Docker Volumes. Also, we wrote scripts to make those operations automatically. Plus, we ran a Jupyter Notebook using the same volume where the data was saved and we implemented some exploratory data analysis. The sensitive data as passwords and tokens were handled safely. After doing everything locally as a Docker Stack, we connected our GitHub repository on the Google Cloud Platform and deployed our solution on the Cloud!

Conclusion

After doing the proposed tutorials during the course classes we could imagine the complexity of Big Data infrastructure. However, after doing the group project, we started to understand such complexity due to the great challenge that was to deploy a simple application both locally and on the Cloud. The use of a lot of structures that characterize Big Data infrastructure solutions can easily become hard work even when we have good tools to help us out.