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)

Contents

- Collisions Routieres Dataset
- Downloading the data
- Creating Volume
- Services on Spark Cluster
- Preparing Data and Work Environment
- Store the data in HDFS and MongoDB
- Automated Storage
- Using Jupyter Notebook
- Google Cloud Platform

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)

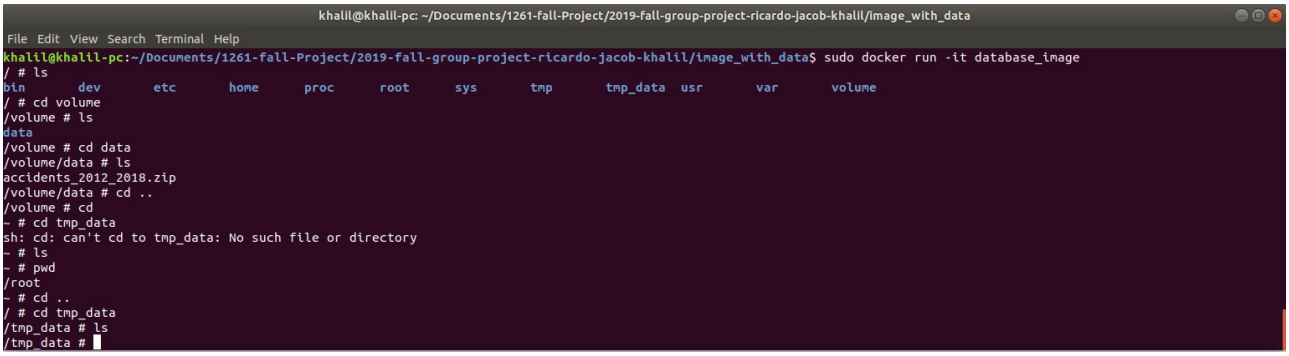
Dataset Characteristics

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) 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
```

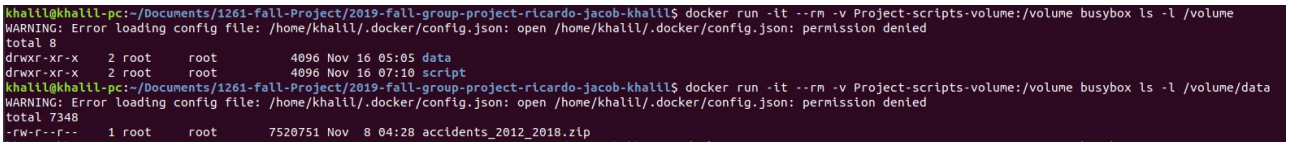
Copy Data Folder to Volume

```
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) file using the below command:

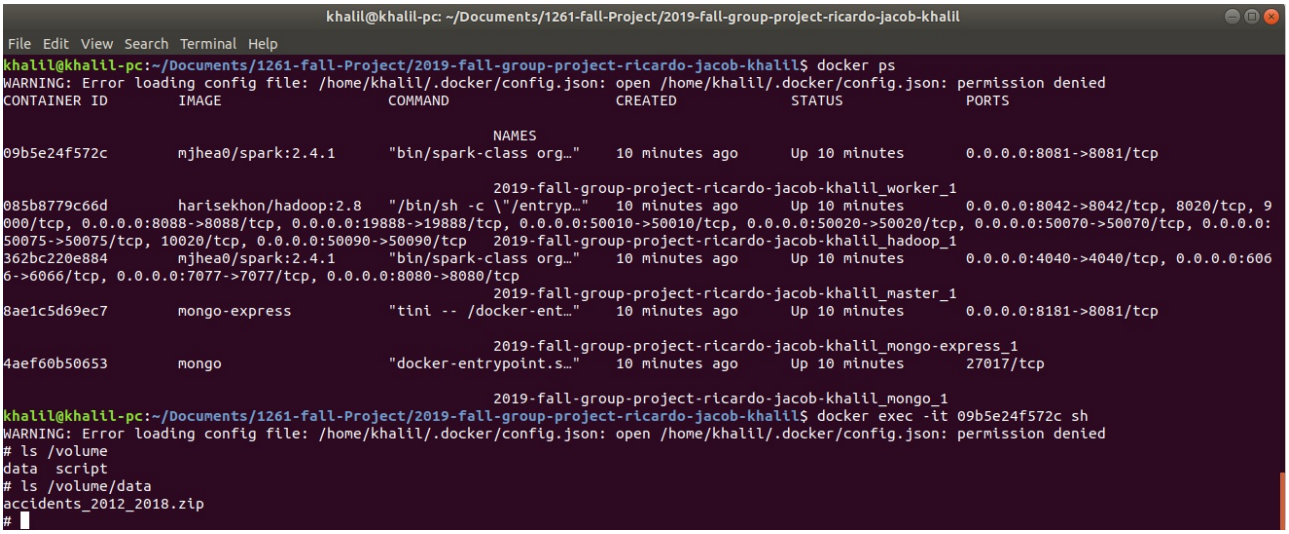
```
env user_mongo=root pass_mongo=password docker-compose --file spark-compose.yml
up --scale spark-worker=2
```

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:

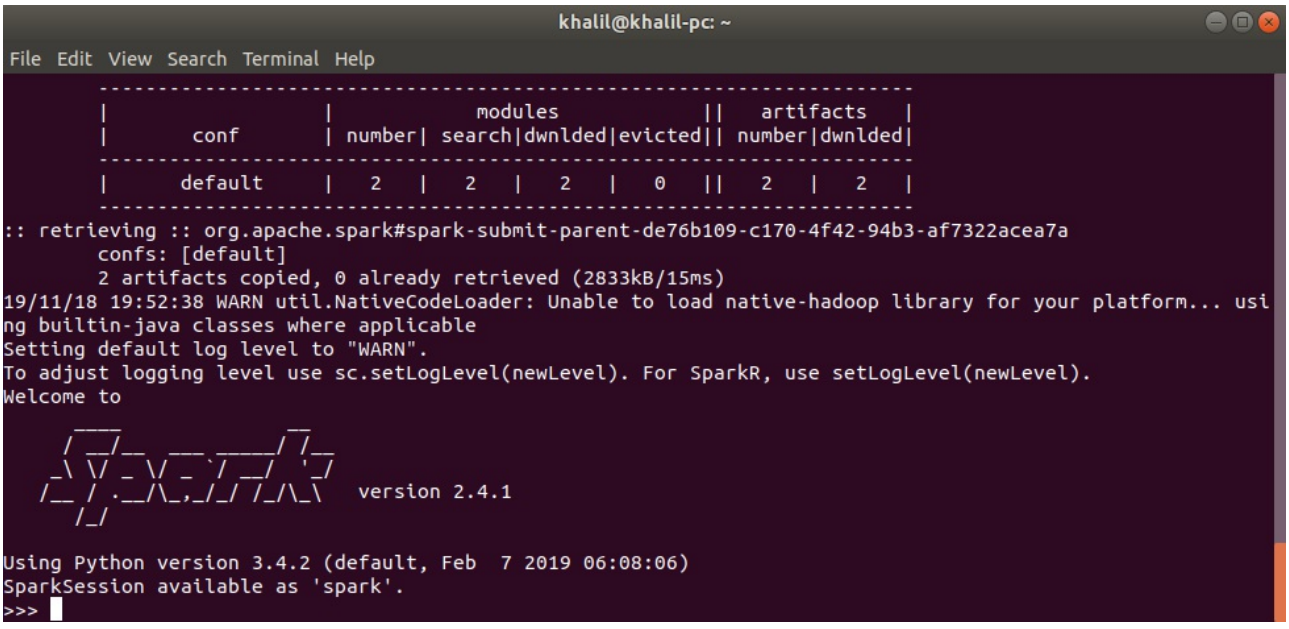
```
docker exec -it containerID sh
```



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
```



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

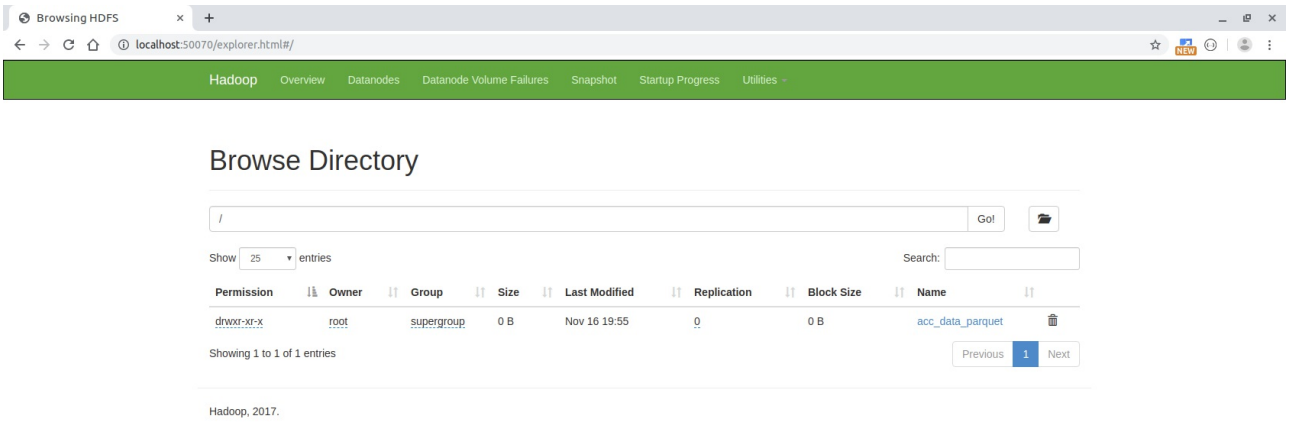
Store the data in HDFS as parquet

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()
```

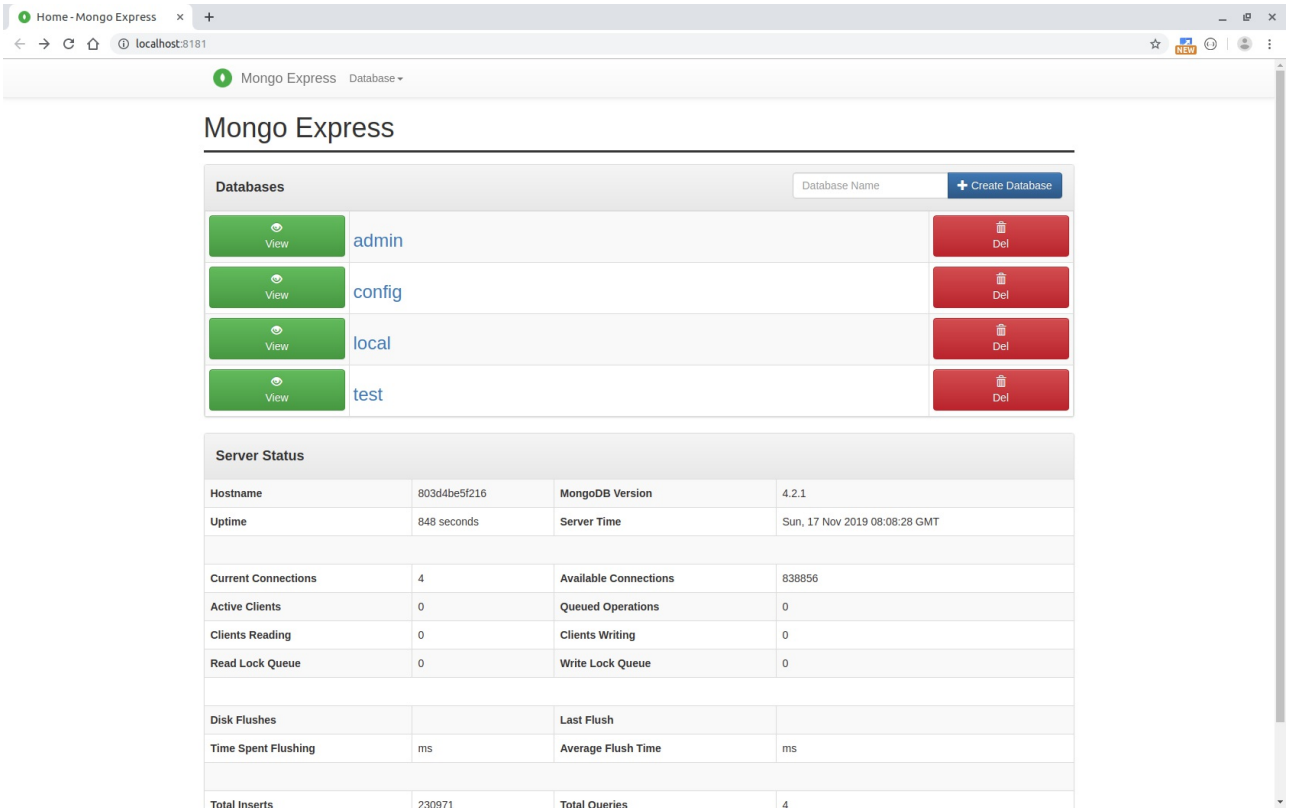
Read the data from the volume and store it in MongoDB

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

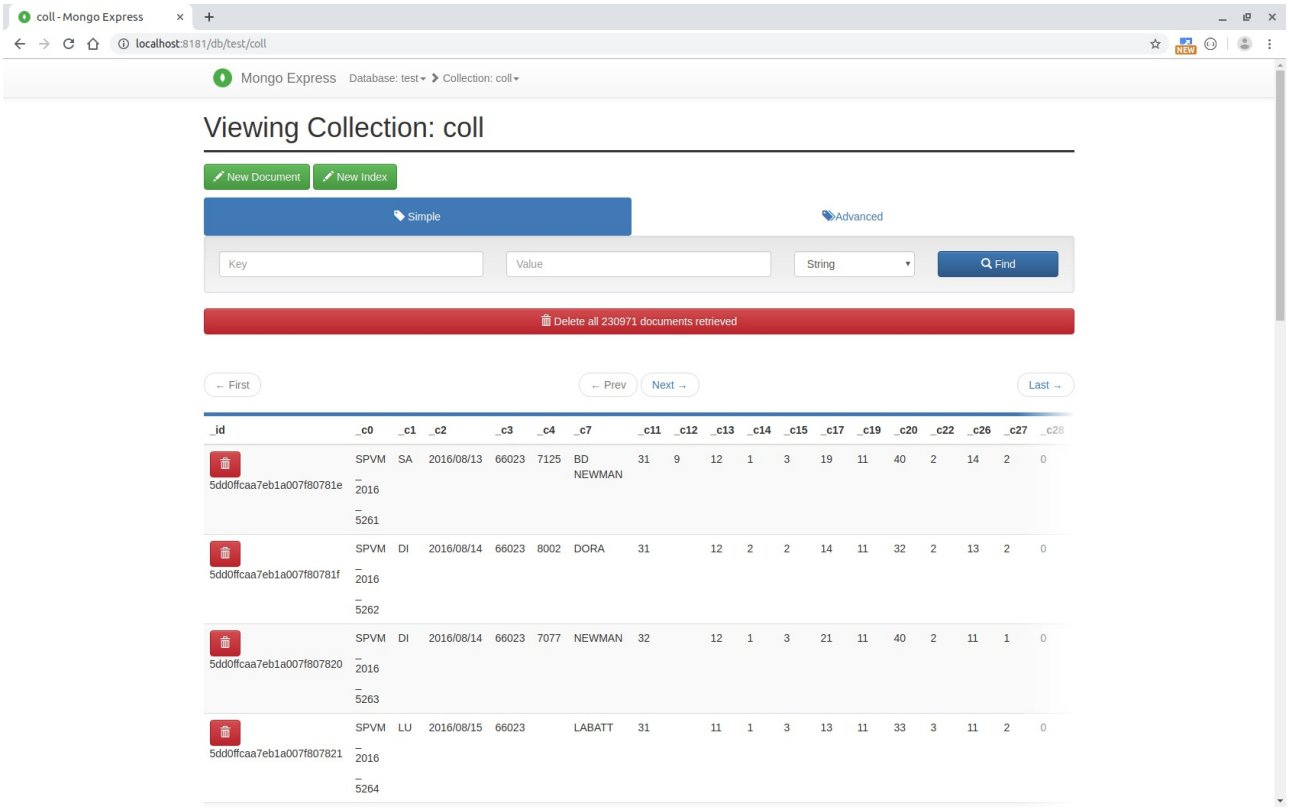
Read the data from the HDFS and store it in MongoDB

```
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)]() 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) 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) 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()
```

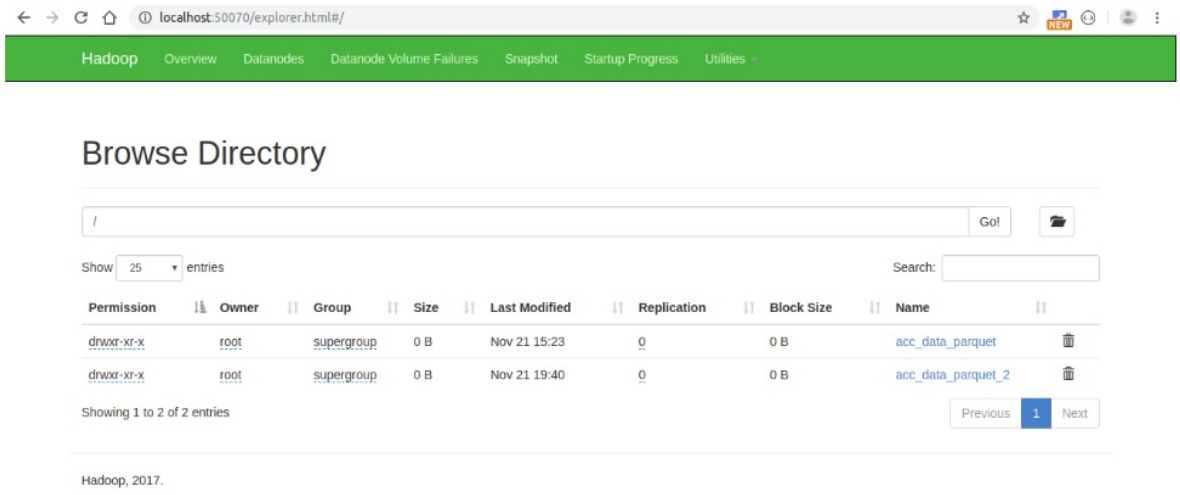
```
root
 |-- _c0: string (nullable = true)
 |-- _c1: string (nullable = true)
 |-- _c2: string (nullable = true)
 |-- _c3: string (nullable = true)
 |-- _c4: string (nullable = true)
 |-- _c5: string (nullable = true)
```

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

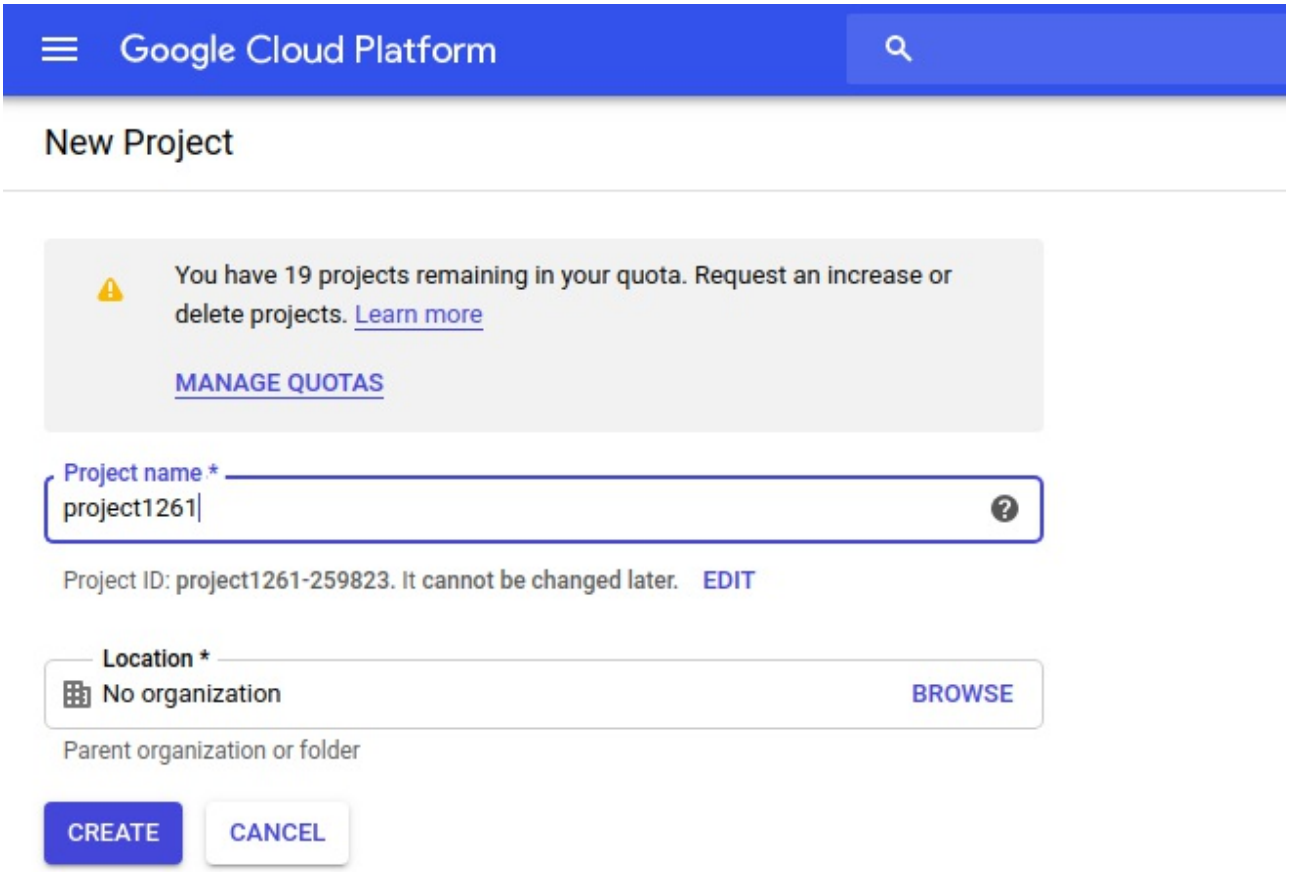
```
In [8]: acc_mongo_jupyter.write.format("com.mongodb.spark.sql.DefaultSource").mode("append").save()
```

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

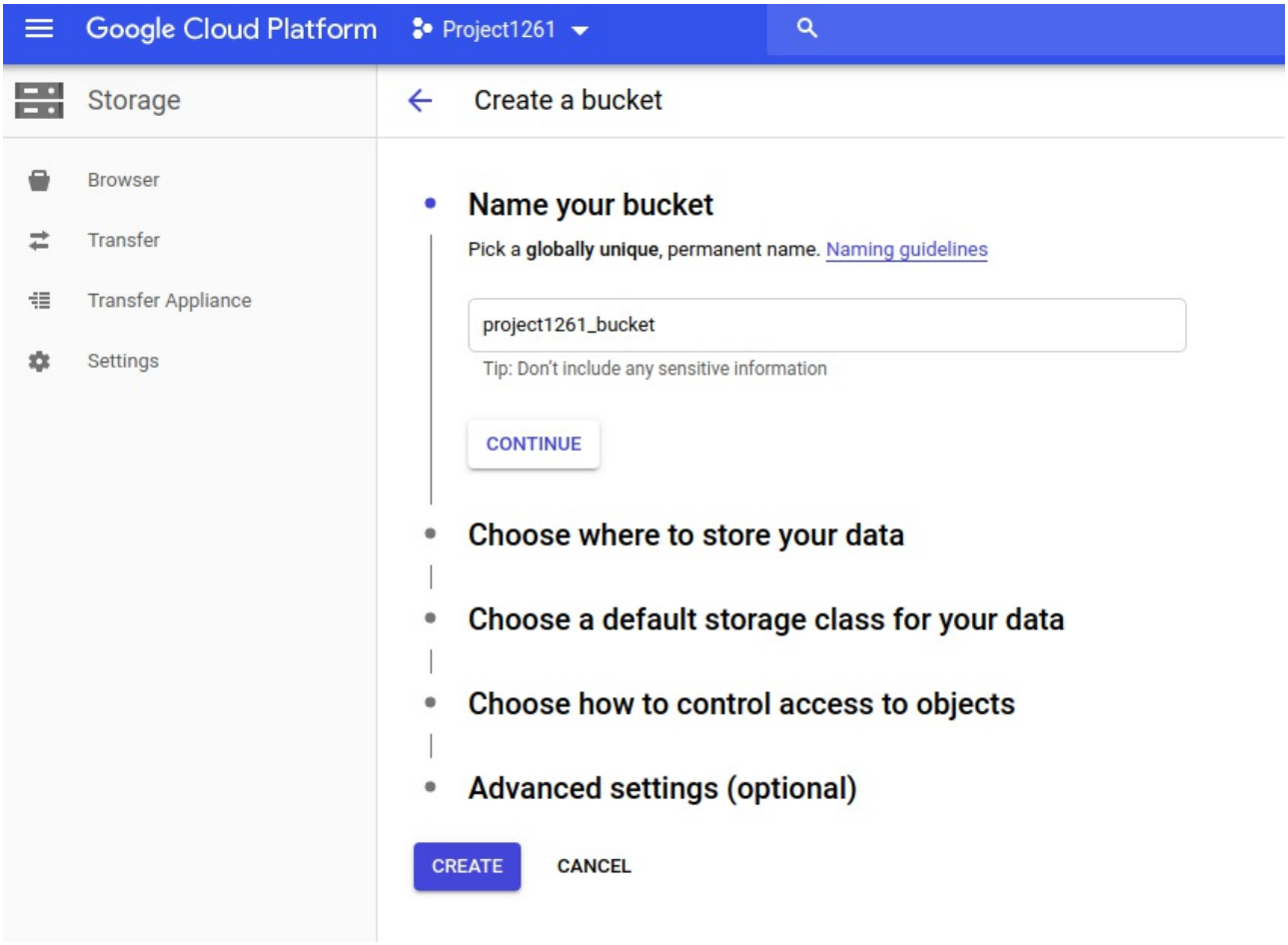
Google Cloud Platform

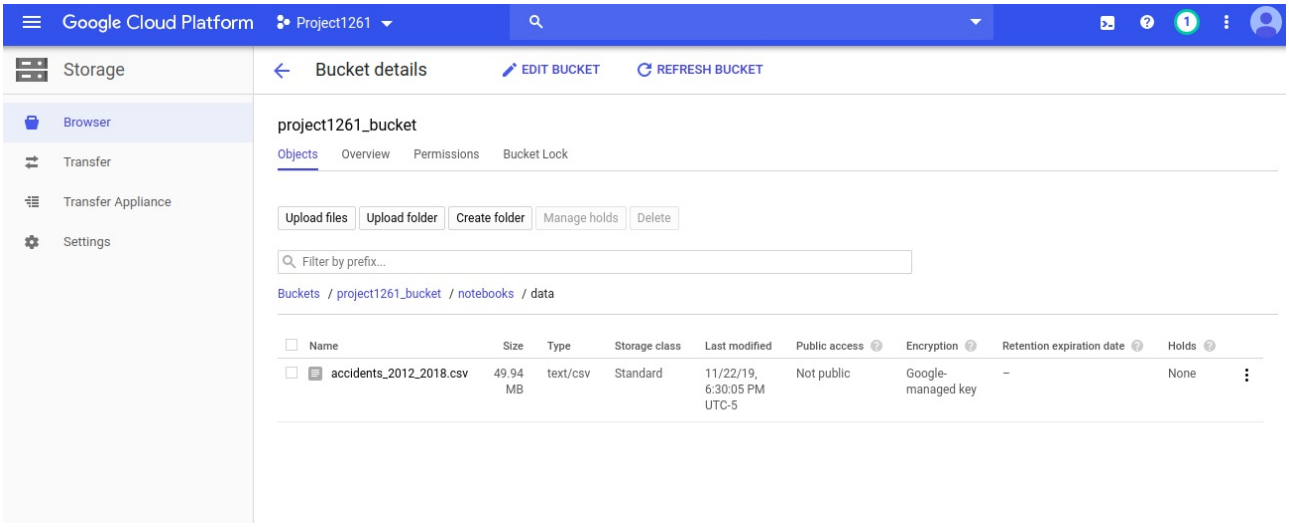
Environment Setup

- First lets create a project.

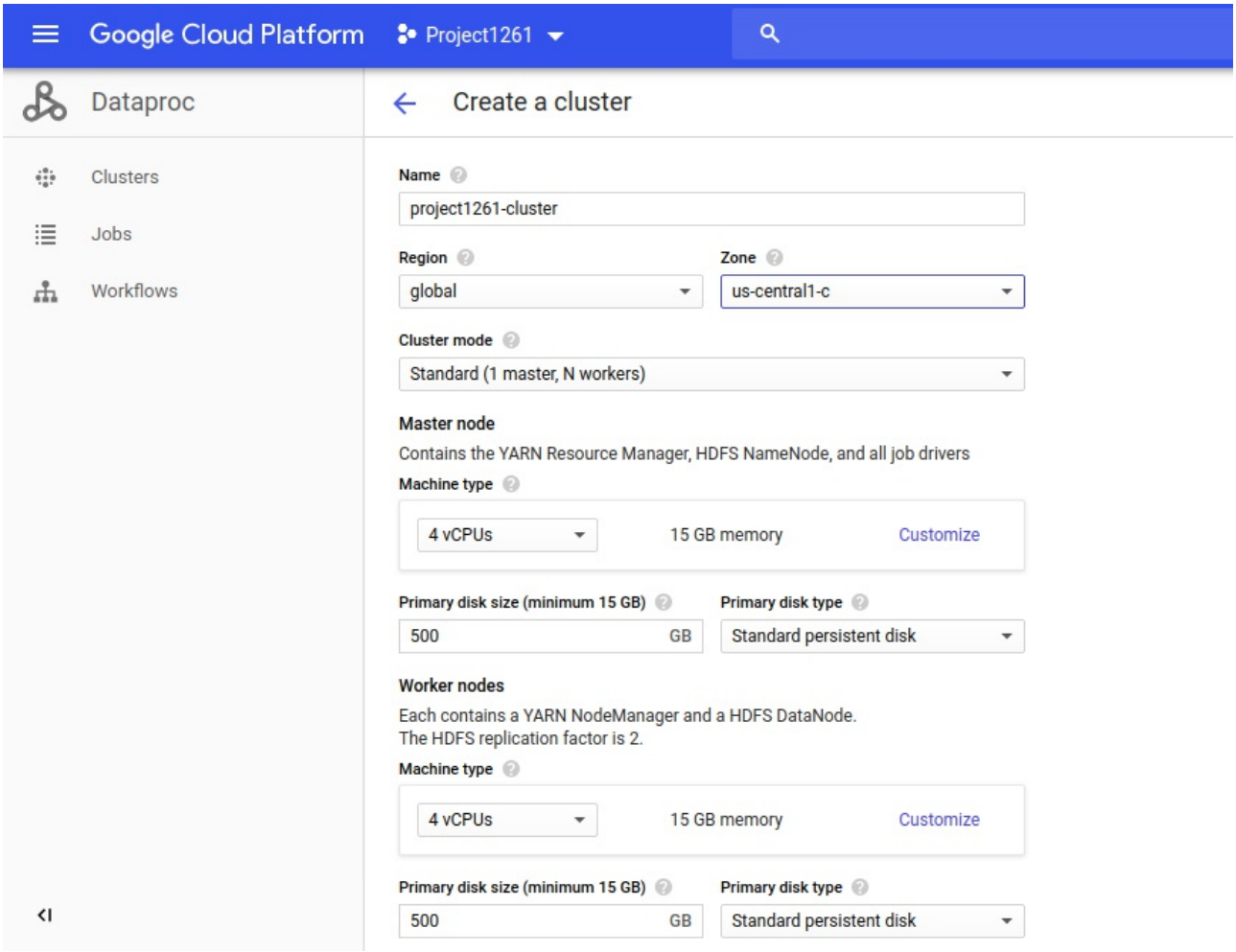


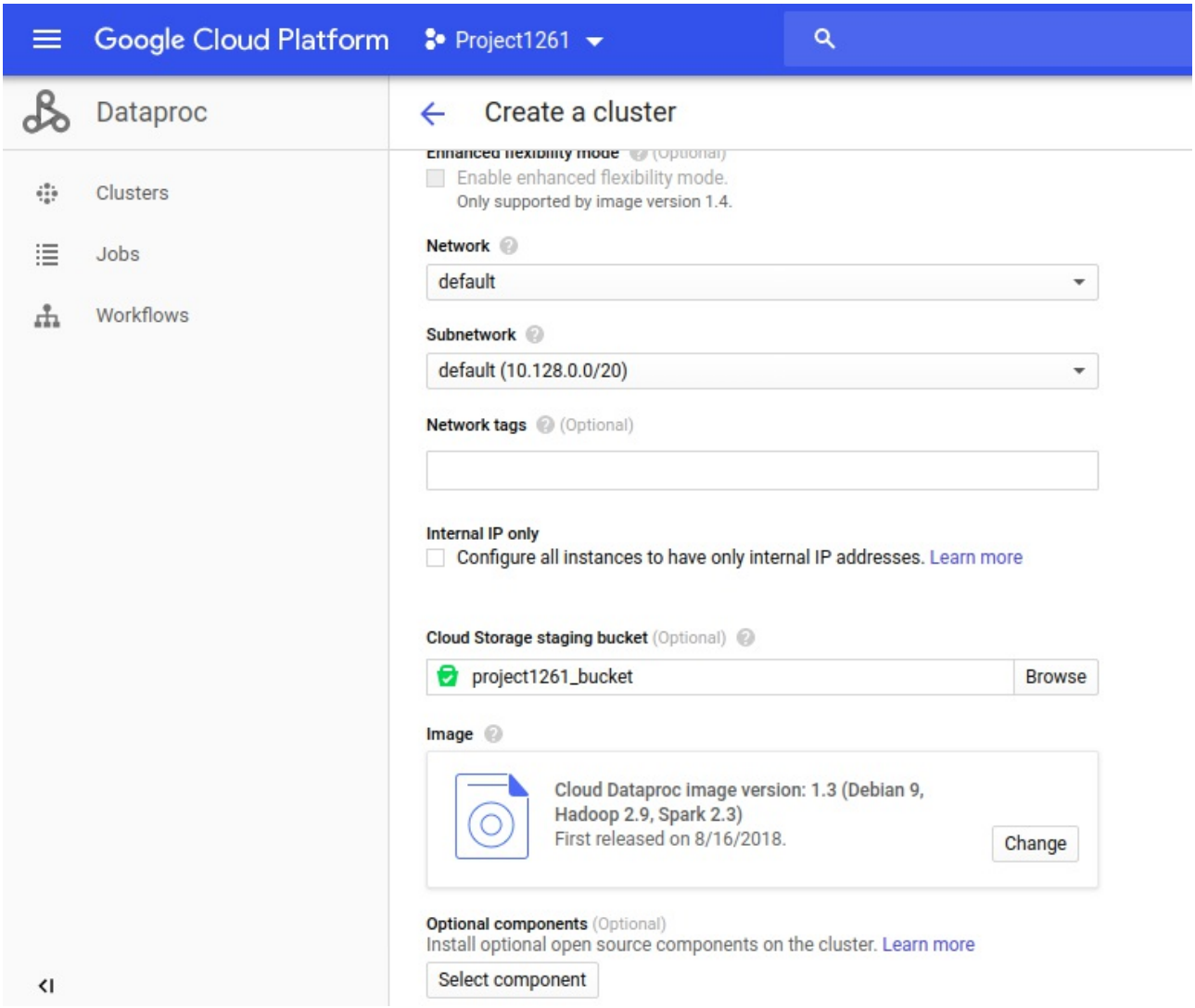
- Enabling the Cloud Dataproc and Google Compute Engine APIs.
- Creating bucket where notebook and the data will be stored.



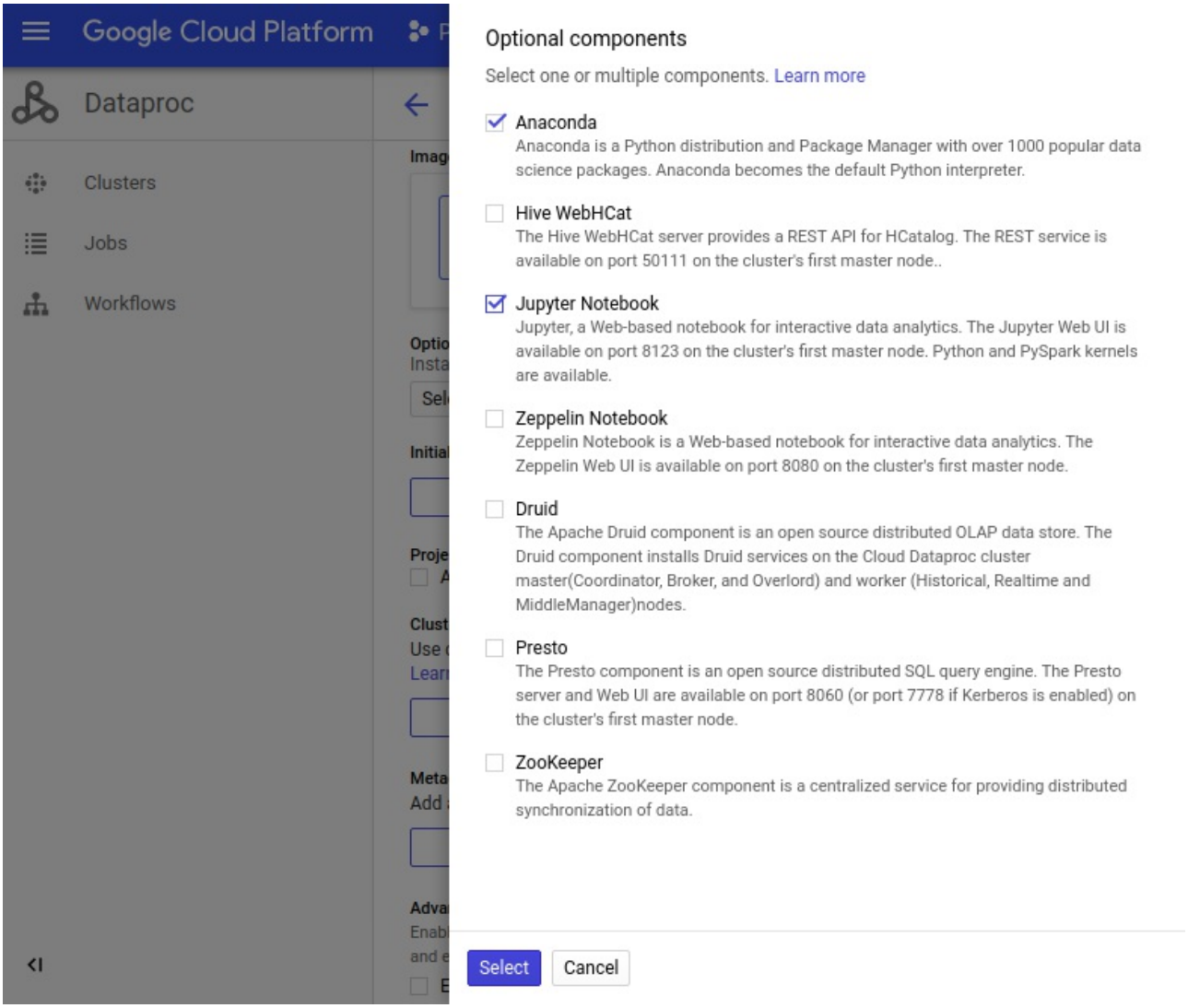


- Creating cluster and selecting the the bucket to which was created to have access to the data and where the notebooks will be stored.

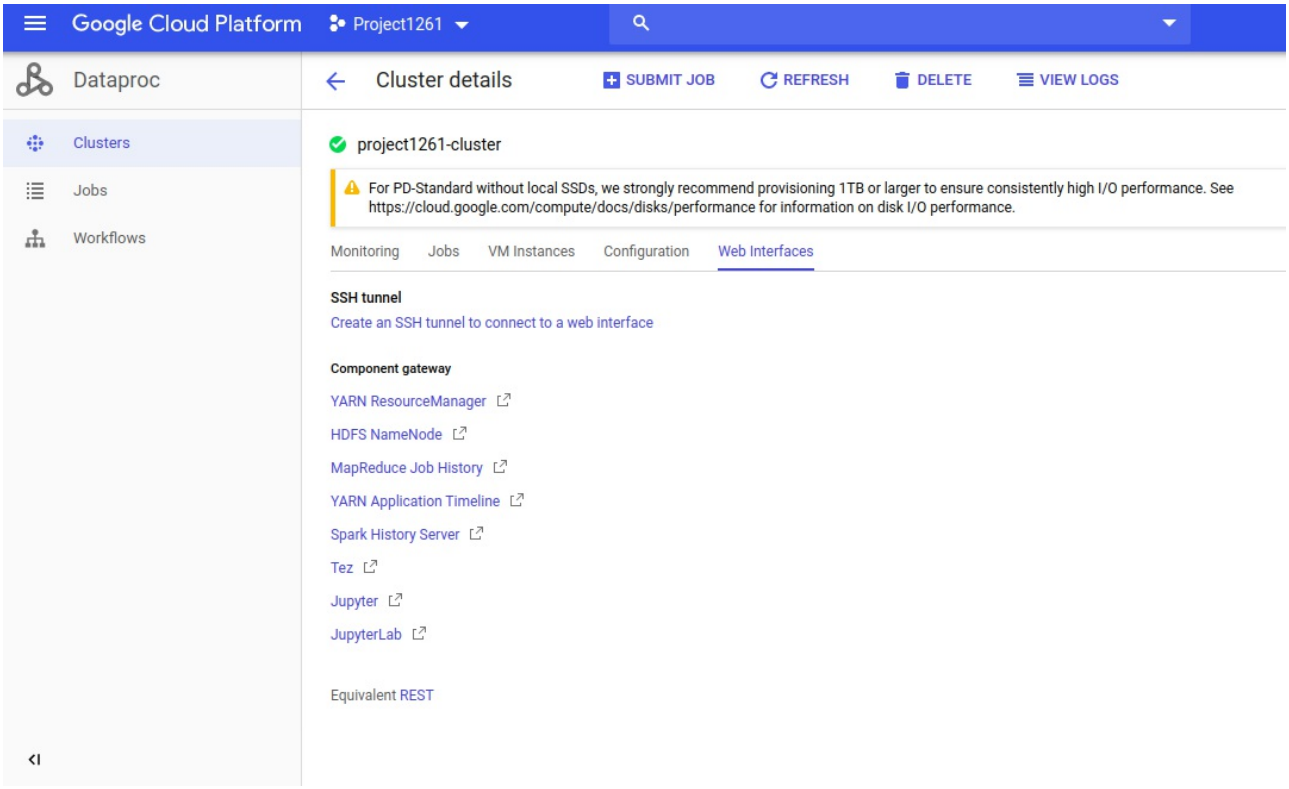

Google Cloud Platform
Project1261
Dataproc
Clusters
Jobs
Workflows
Create a cluster
Name
project1261-cluster
Region
global
Zone
us-central1-c
Cluster mode
Standard (1 master, N workers)
Master node
Contains the YARN Resource Manager, HDFS NameNode, and all job drivers
Machine type
4 vCPUs
15 GB memory
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.

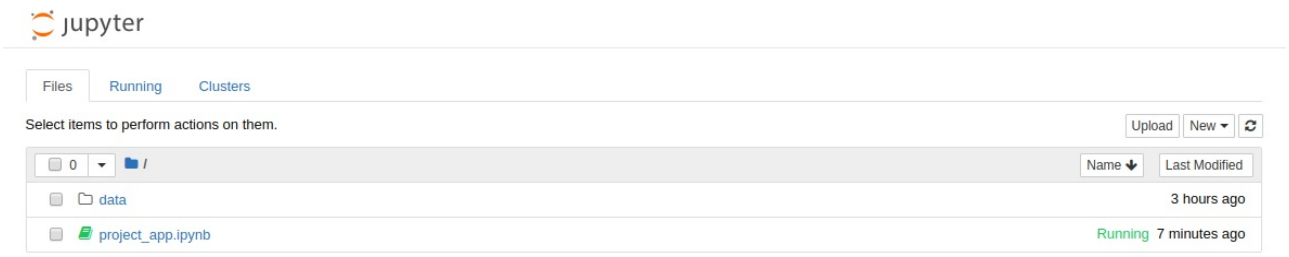


- From the web interfaces in created cluster we can open jupyter notebook which is runing on the cluster and have access to the bucket where our data is.



Analytics on GCP

Openning Jupyter on GCP cluster

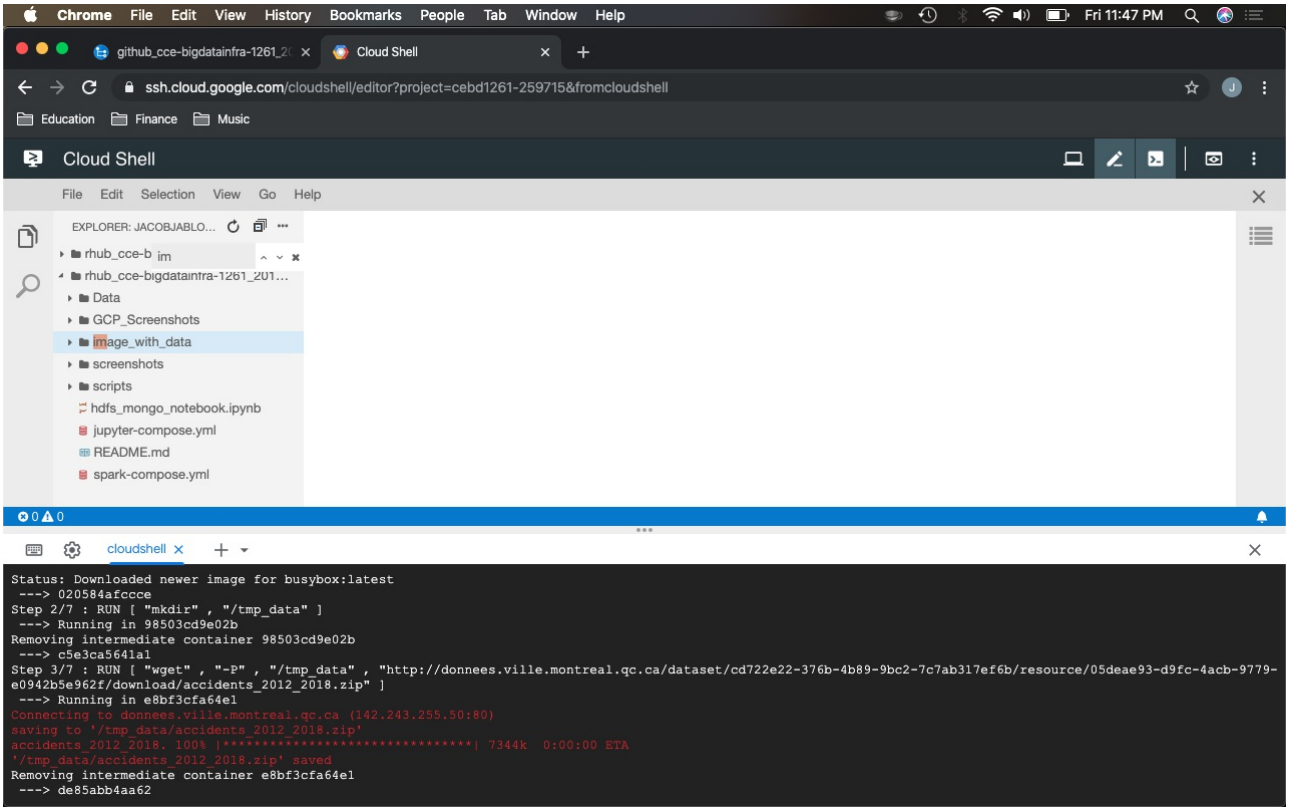


Running the Project App
[The App Notebook (./project_app.ipynb)]

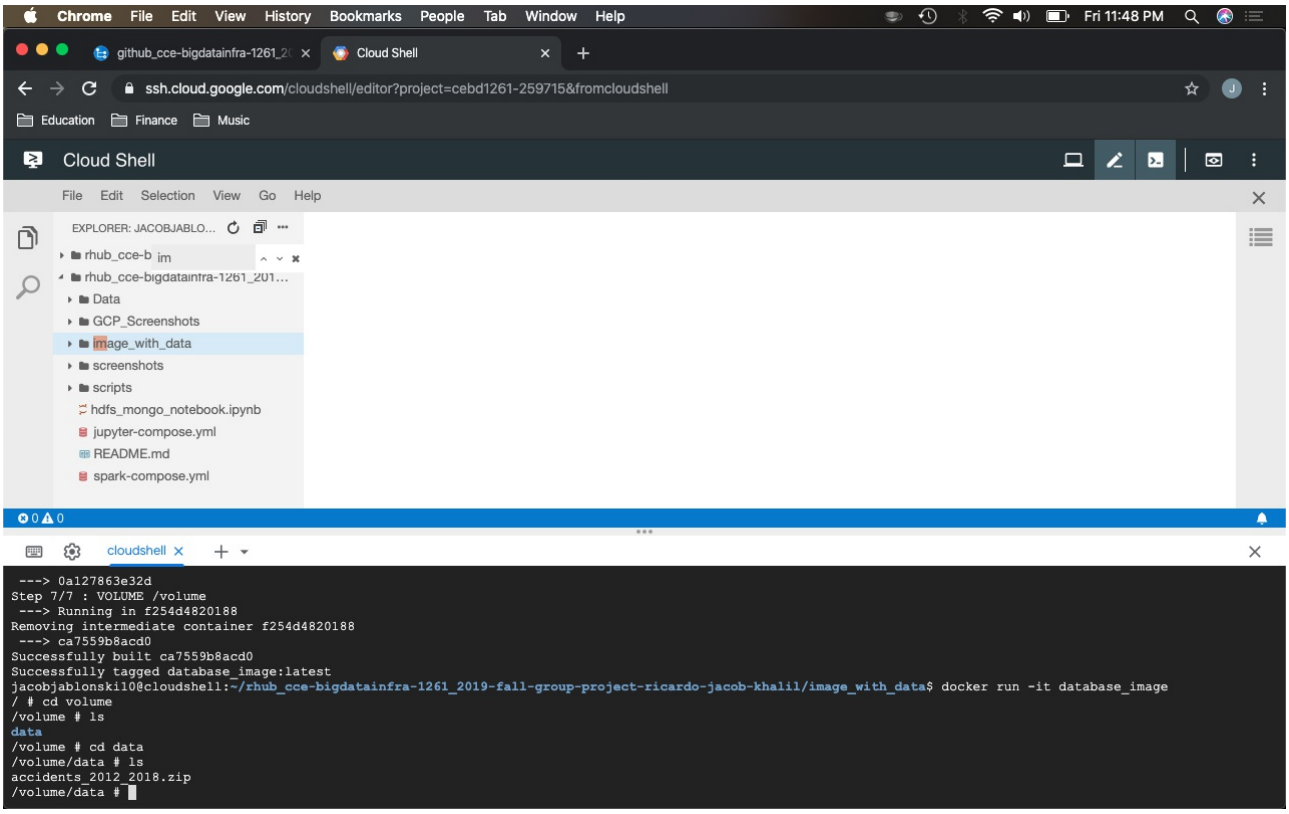
Github Mirror and Deploy via GCP

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.

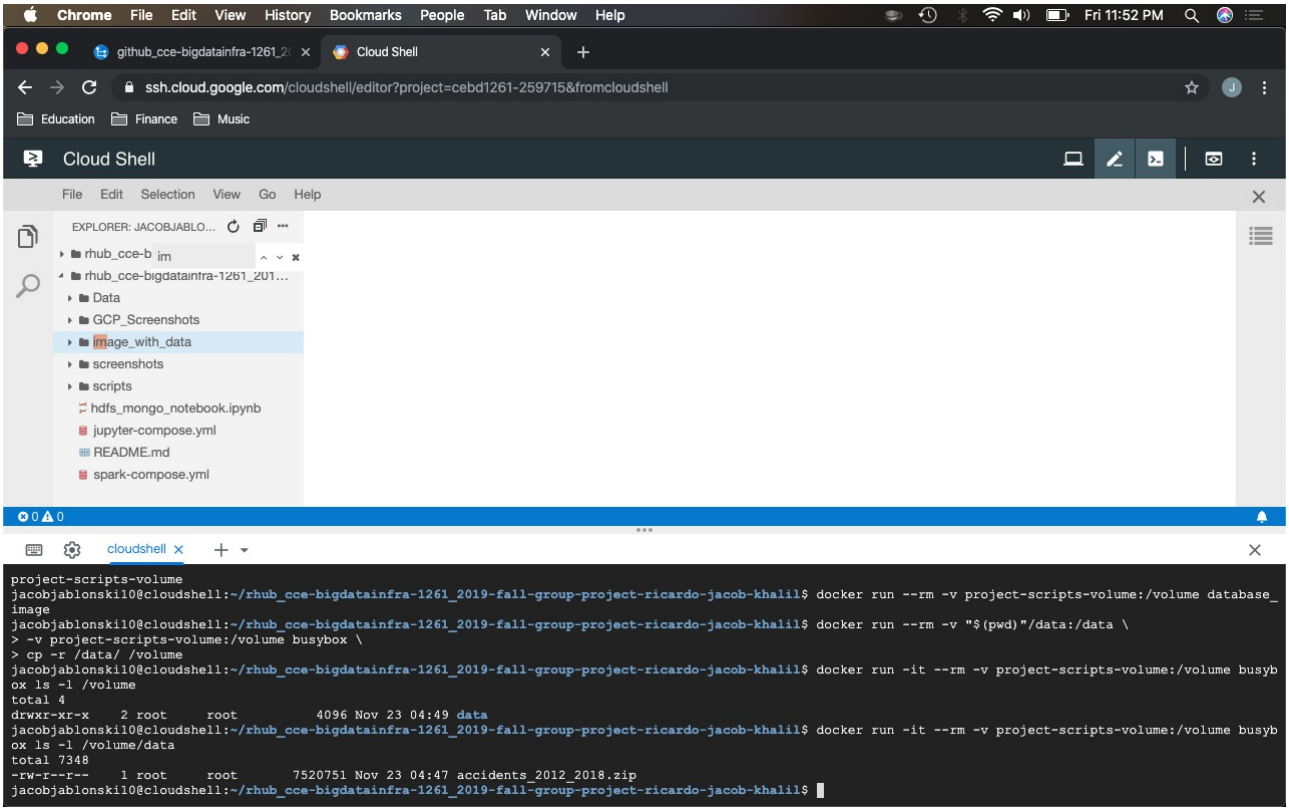
- Pulling data from github hosted docker image through google shell



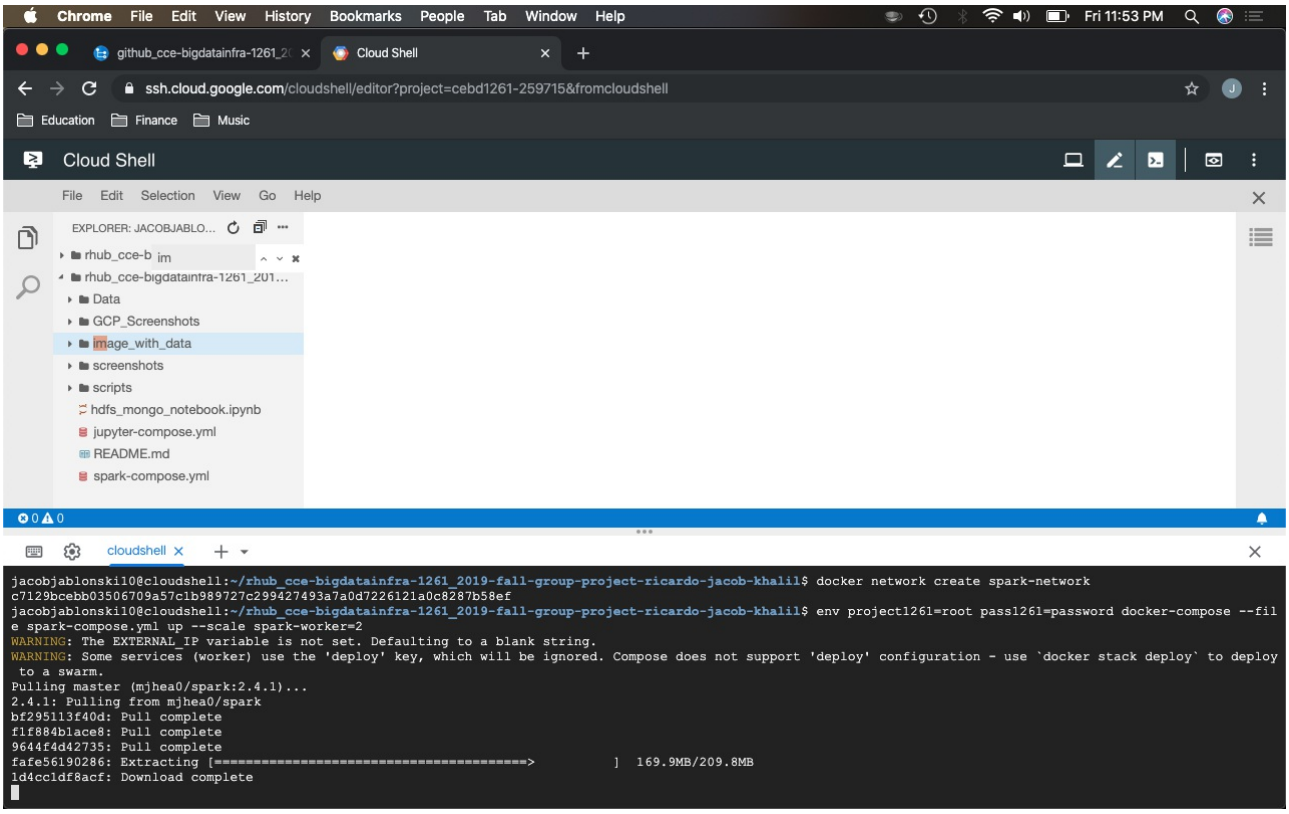
- Running the docker image and making sure the data is in the correct directory



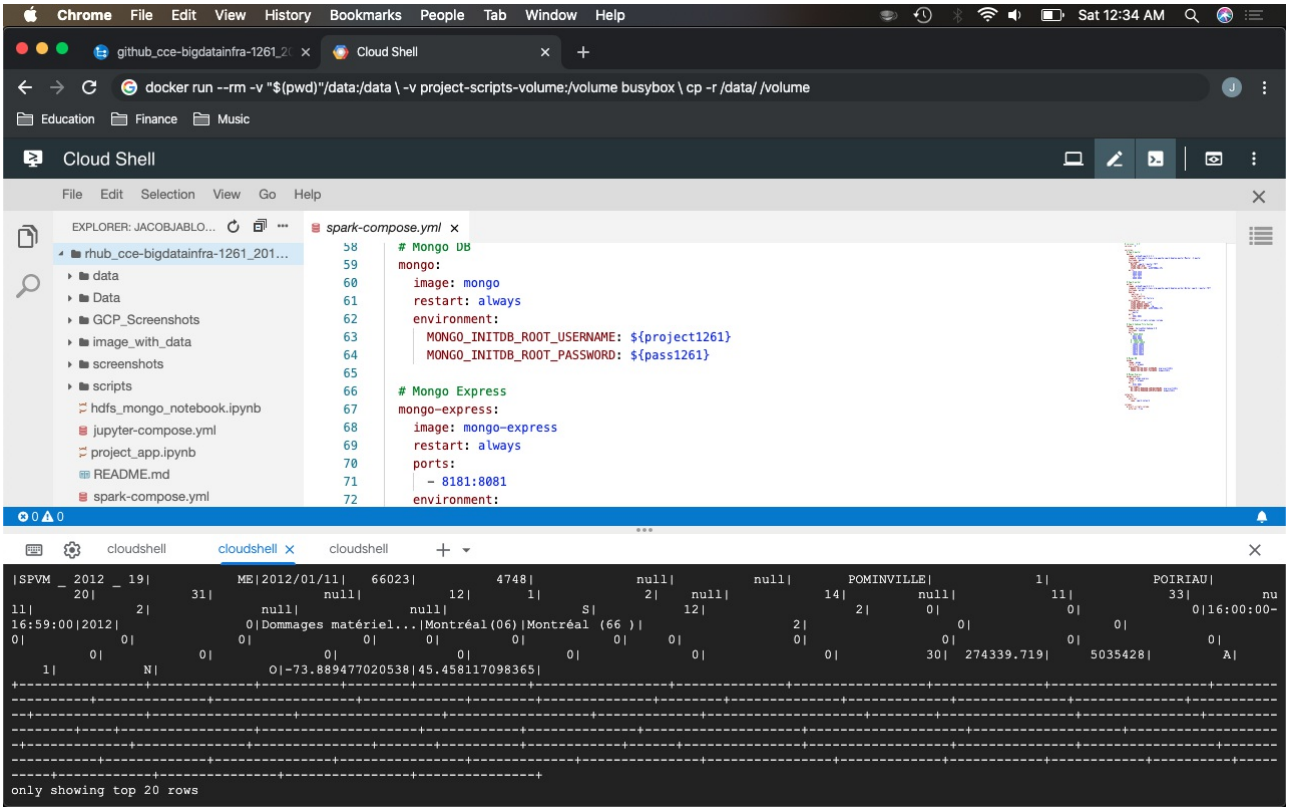
- Copying the scripts



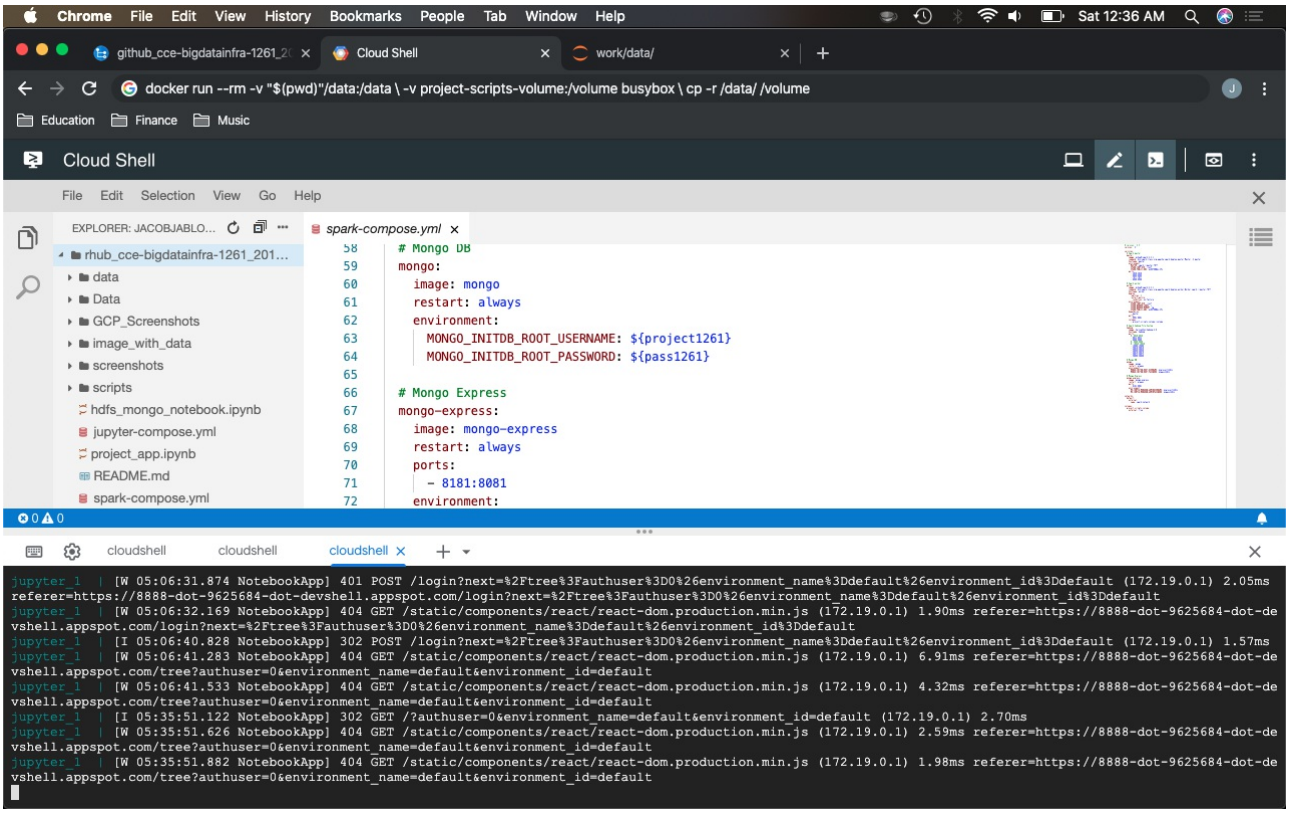
- Deploying the spark compose file with created network



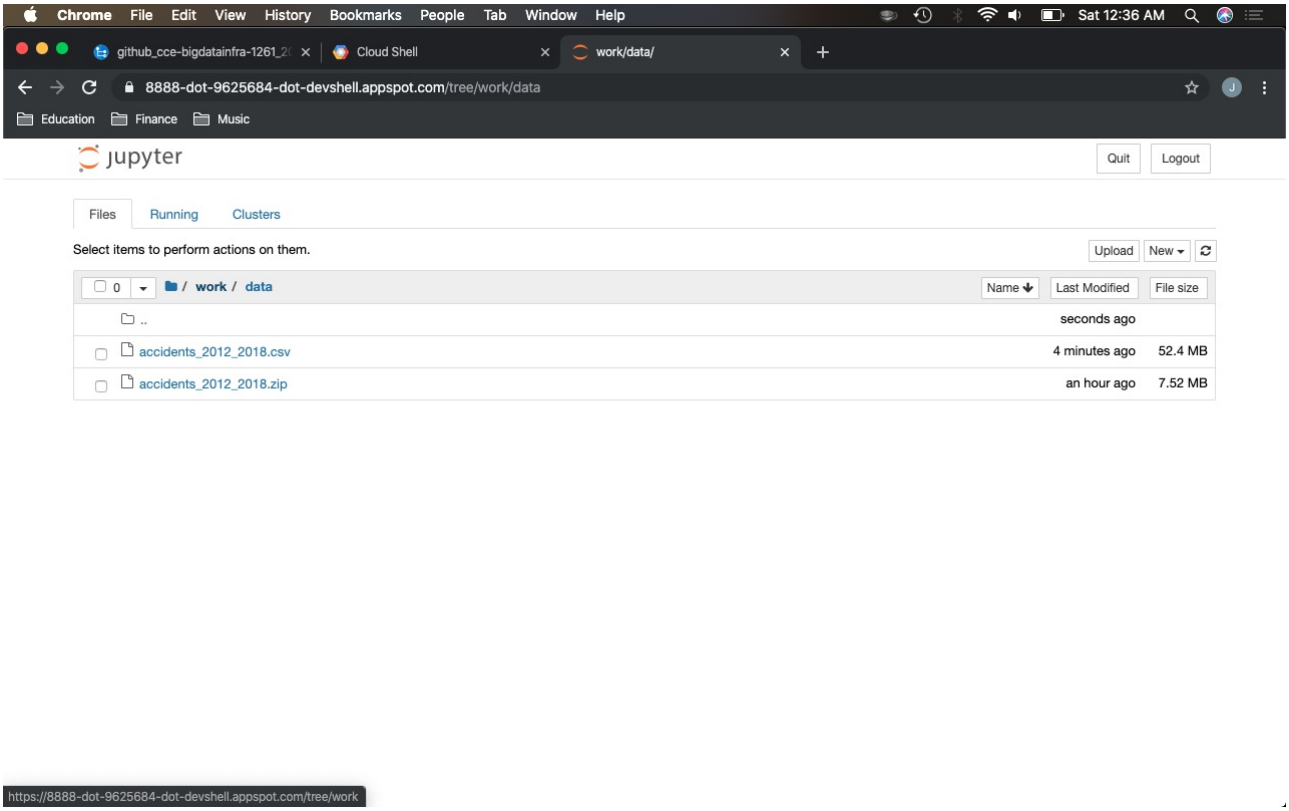
- Creating jupyter volume



- Jupyter compose up to be able to reach our notebook



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


Chrome File Edit View History Bookmarks People Tab Window Help
Sat 12:36 AM
github_cce-bigdatainfra-1261_2
Cloud Shell
work/data/
8888-dot-9625684-dot-devshell.appspot.com/tree/work/data
Education
Finance
Music
jupyter
Quit
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
0
/ work / data
Name
Last Modified
File size
..
seconds ago
accidents_2012_2018.csv
4 minutes ago
52.4 MB
accidents_2012_2018.zip
an hour ago
7.52 MB
https://8888-dot-9625684-dot-devshell.appspot.com/tree/work