CEBD 1261 Final Project

Prepared by:
Ieva Betina
William Acosta

Table of Contents

- Create the cluster using Docker Swarm
- Read and save the data (HDFS, Parquet files, MongoDB)
- Load the data with Jupyter notebook from MongoDB
- Security precautions with environment variables
- Analyze the data with Spark Dataframes
- Spark with Google Cloud Platform
- Future improvements
- Conclusion

Summary

This project will demonstrate running a Spark Cluster with Docker Swarm on a single node and also on Google Cloud Platform. As part of the demonstration we will read a London energy csv with Spark and write it to HDFS using Parquet files and then also to MongoDB. From there, we will use Jupyter notebook as a front end to analyze the data. On GCP, we will demonstrate reading the csv from a bucket and saving back to the bucket in the Parquet format. Jupyter notebook will again be used to analyze the data.

Create a Spark Cluster

NOTE

- make sure you have enough Docker resources. Increase the memory and CPUs.
- for this project we'll be running the swarm on just one node. We could have also run these services with Docker compose since there is only one host.

Get a swarm up and running:

```
docker swarm init
```

For multi-node clusters, you might have to use a specific hostname:

```
docker swarm init --advertise-addr $(hostname -i)
```

Create a shared overlay network named "spark-network":

```
docker network create -d overlay --attachable spark-network
```

This network allows the containers to communicate and can be added to any new containers outside of the swarm to communicate with the containers in the swarm.

Create a volume for files

We use a named volume "london-power" for storing files. We will be referencing this volume in our stack.yml so that the services can connect to it.

```
docker volume create london-power
```

ALTERNATIVELY:

Don't create the volume since the yml file can create it for you at launch.

Create a custom docker image

We modified the mjhea0/spark:2.4.1 image to already include our python scripts. We created a simple Dockerfile with the following:

```
FROM mjhea0/spark:2.4.1
COPY script/ /usr/spark-2.4.1/script
```

Then we build the image. We added the file to docker hub so we included a user name and tags:

```
docker build -t wacosta/spark-custom:2.4.1 .
```

Push to docker hub:

```
docker push wacosta/spark-custom:2.4.1
```

The script files will now be located in the workers when the stack launches.

Alternatives

Getting a file directly from a website either with `ADD` or `RUN wget` then unzipping them with `RUN unzip` would be a great way to automate updating of the files by using this custom Docker image when deploying our stack. Just exploring that it is possible for educational purposes. However, the documentation recommends against it and large files should be put directly in a volume :

```
FROM mjhea0/spark:2.4.1

RUN apt-get update

RUN apt-get install -y wget

RUN mkdir /mongo

USER root

RUN wget "https://data.london.gov.uk/download/smartmeter-energy-use-data-in-london-households/3527bf39-d93e-4071-8451-df2ade1ea4f2/Power-Networks-LCL-June2015.zip" -P  /usr/spark-2.4.1/volume/

RUN unzip /usr/spark-2.4.1/volume/Power-Networks-LCL-June2015.zip
```

We can see that the file has unzipped and is inside the spark worker:

```
cce-fb111-02m:2019-spring-group-project-ieva-william i_beti$  docker exec -it aa08b39161f8   sh
# ls
LICENSE  NOTICE  Power-Networks-LCL-June2015(withAcornGps)v2.csv  R  README.md  RELEASE  bin  conf  data  examples  jars  kubernetes  licenses  python  sbin  volume  work  yarn
#
```

Another interesting option is to add files to the image from the cloud using AWS s3 bucket or gcp bucket. It would be worth exploring.

Prepare the stack yml file

The stack-spark.yml file must include all the services necessary for this project. This includes:

- spark master
- spark worker (using custom image)
- hadoop

- jupyter notebook
- mongoDB

It is also necessary to stipulate the named volume "london-power" that some of the containers will be using as well as the overlay network "spark-network".

```
    volumes:
      - "london-power:/usr/spark-2.4.1/volume"
    # AND add this at the end (external necessary for existing volumes)
volumes:
  london-power:
    external: true

networks:
  default:
    external:
      name: spark-network
```

Set your environment variables before deploying

The mongo and jupyter services make use of environment variables in the yml file. These will be discussed in the environment variables section below.

```
export ME_CONFIG_MONGODB_ADMINUSERNAME=root
export ME_CONFIG_MONGODB_ADMINPASSWORD=example
export MONGO_INITDB_ROOT_USERNAME=root
export MONGO_INITDB_ROOT_PASSWORD=example
export JUPYTER_TOKEN='project'
```

Deploy the stack

```
docker stack deploy --compose-file=stack-spark.yml spark
```

If necessary scale the workers up or down. The default we set is 3:

```
docker service scale spark_worker=5
```

Populate the volume

Use the busybox hack to copy files to a volume

```
docker run --rm -v "$(pwd)"/data:/data -v london-power:/volume busybox cp -r data/ /volume
```

If you don't write scripts directly into the image, add them this way:

```
docker run --rm -v "$(pwd)"/script:/script -v london-power:/volume busybox cp -r script/ /volume
```

Look inside the volume

Use the busybox hack to peak inside and see the files

```
docker run -it --rm -v london-power:/vol busybox sh
```

NOTE:

If using the busybox hack, the volume must be populated *after* the stack is deployed. Otherwise, permission in the jupyter notebook will be denied when trying to start a new notebook.

Read and save the data (HDFS, Parquet files, MongoDB)

Run the .py file with spark-submit to save csv to HDFS and MongoDB

The [csv-hdfs-mongo.py] reads the csv as a dataframe in Spark. Then it gives it a schema and the proper headers which are specified by us. It saves the dataframe to HDFS as a parquet file. The parquet file is then loaded as a dataframe and saved to MongoDB. This is all done using spark submit. The mongo input and output uri are specified with environment variables for security.

We wrote the files to HDFS *and* MongoDB for demonstration purposes. We would use one or the other for regular use. We've also included other reference scripts for the following:

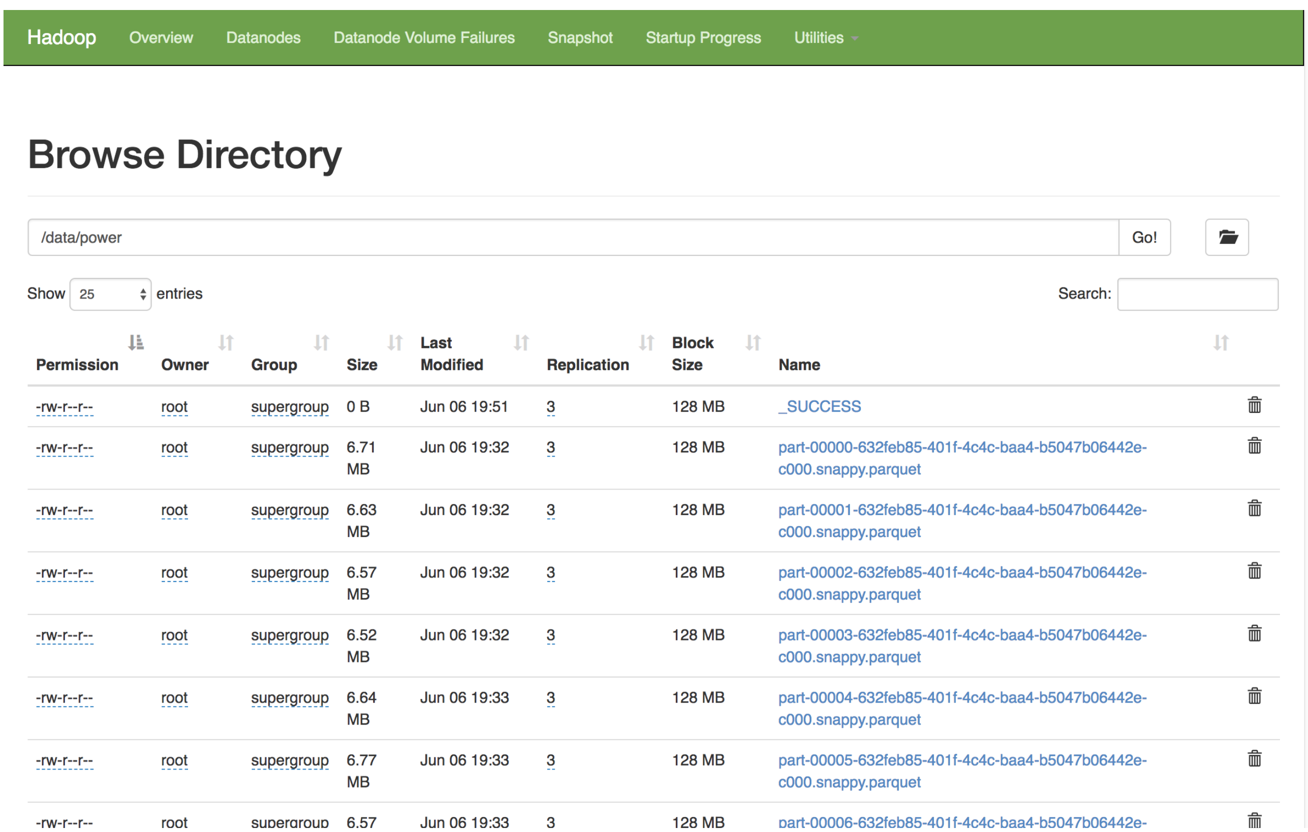
- csv to MongoDB
- csv to hdfs
- hdfs to MongoDB

Note that the spark submit needs the MongoDB connector passed as a package as well as the input/output uri passed with a --conf flag.

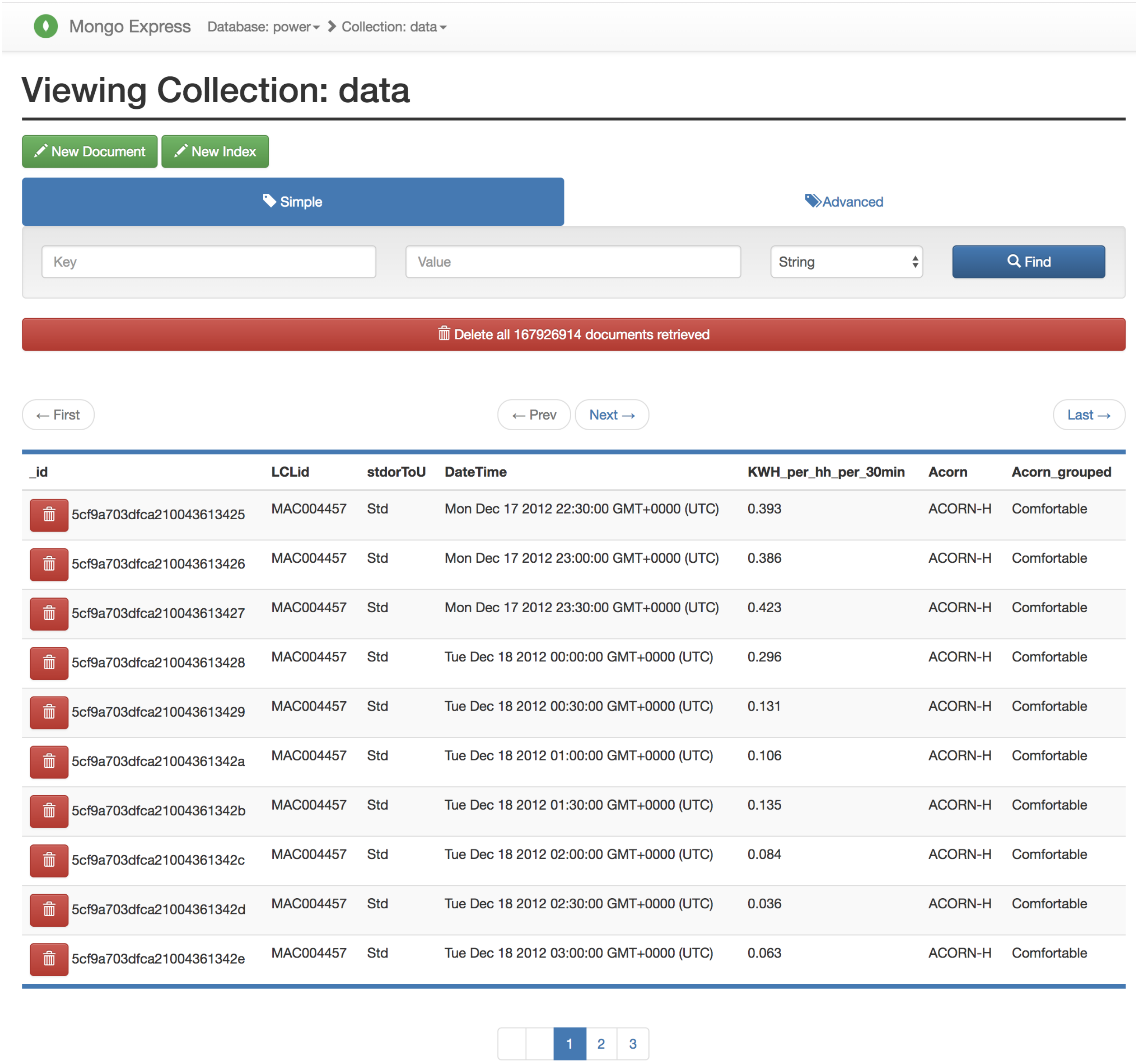
```
docker run -t --rm \
  -v london-power:/usr/spark-2.4.1/volume \
  --network=spark-network \
  wacosta/spark-custom:2.4.1 \
  bin/spark-submit \
    --master spark://master:7077 \
    --class endpoint \
    --packages org.mongodb.spark:mongo-spark-connector_2.11:2.4.0 \
    --conf "spark.mongodb.input.uri=mongodb://"$MONGO_INITDB_ROOT_USERNAME:$MONGO_INITDB_ROOT_PASSWORD"@mongo/power.data?
authSource=admin" \
    --conf "spark.mongodb.output.uri=mongodb://"$MONGO_INITDB_ROOT_USERNAME:$MONGO_INITDB_ROOT_PASSWORD"@mongo/power.data?
authSource=admin" \
    /usr/spark-2.4.1/script/csv-hdfs-mongo.py
```

View the results in HDFS at http://localhost:50070

View the results in MongoDB at http://localhost:8181/



Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	root	supergroup	0 B	Jun 06 19:51	3	128 MB	_SUCCESS
-rw-r--r--	root	supergroup	6.71 MB	Jun 06 19:32	3	128 MB	part-00000-632feb85-401f-4c4c-baa4-b5047b06442e-c000.snappy.parquet
-rw-r--r--	root	supergroup	6.63 MB	Jun 06 19:32	3	128 MB	part-00001-632feb85-401f-4c4c-baa4-b5047b06442e-c000.snappy.parquet
-rw-r--r--	root	supergroup	6.57 MB	Jun 06 19:32	3	128 MB	part-00002-632feb85-401f-4c4c-baa4-b5047b06442e-c000.snappy.parquet
-rw-r--r--	root	supergroup	6.52 MB	Jun 06 19:32	3	128 MB	part-00003-632feb85-401f-4c4c-baa4-b5047b06442e-c000.snappy.parquet
-rw-r--r--	root	supergroup	6.64 MB	Jun 06 19:33	3	128 MB	part-00004-632feb85-401f-4c4c-baa4-b5047b06442e-c000.snappy.parquet
-rw-r--r--	root	supergroup	6.77 MB	Jun 06 19:33	3	128 MB	part-00005-632feb85-401f-4c4c-baa4-b5047b06442e-c000.snappy.parquet
-rw-r--r--	root	supergroup	6.57	Jun 06 19:33	3	128 MB	part-00006-632feb85-401f-4c4c-baa4-b5047b06442e-



_id	LCLid	stdorToU	DateTime	KWH_per_hh_per_30min	Acorn	Acorn_grouped
5cf9a703dfca210043613425	MAC004457	Std	Mon Dec 17 2012 22:30:00 GMT+0000 (UTC)	0.393	ACORN-H	Comfortable
5cf9a703dfca210043613426	MAC004457	Std	Mon Dec 17 2012 23:00:00 GMT+0000 (UTC)	0.386	ACORN-H	Comfortable
5cf9a703dfca210043613427	MAC004457	Std	Mon Dec 17 2012 23:30:00 GMT+0000 (UTC)	0.423	ACORN-H	Comfortable
5cf9a703dfca210043613428	MAC004457	Std	Tue Dec 18 2012 00:00:00 GMT+0000 (UTC)	0.296	ACORN-H	Comfortable
5cf9a703dfca210043613429	MAC004457	Std	Tue Dec 18 2012 00:30:00 GMT+0000 (UTC)	0.131	ACORN-H	Comfortable
5cf9a703dfca21004361342a	MAC004457	Std	Tue Dec 18 2012 01:00:00 GMT+0000 (UTC)	0.106	ACORN-H	Comfortable
5cf9a703dfca21004361342b	MAC004457	Std	Tue Dec 18 2012 01:30:00 GMT+0000 (UTC)	0.135	ACORN-H	Comfortable
5cf9a703dfca21004361342c	MAC004457	Std	Tue Dec 18 2012 02:00:00 GMT+0000 (UTC)	0.084	ACORN-H	Comfortable
5cf9a703dfca21004361342d	MAC004457	Std	Tue Dec 18 2012 02:30:00 GMT+0000 (UTC)	0.036	ACORN-H	Comfortable
5cf9a703dfca21004361342e	MAC004457	Std	Tue Dec 18 2012 03:00:00 GMT+0000 (UTC)	0.063	ACORN-H	Comfortable

Remove files from HDFS

If desired, you can remove files from HDFS:

```
docker exec -it $(docker ps --filter name=master --format "{{.ID}}") hdfs dfs -rm -r hdfs://hadoop:8020/data/power/
```

For processing the full 11GB file we used 5 workers using 1 core each and 2gb of memory each. The results were much faster after scaling up for the simple process of reading, setting a schema and writing the file. Notice that before scaling up, the process took 40 minutes on 2 cores and 1gb memory each.

Spark Master at spark://master:7077

URL: spark://master:7077
Alive Workers: 2
Cores in use: 2 Total, 0 Used
Memory in use: 2.0 GB Total, 0.0 B Used
Applications: 0 Running, 1 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (2)

Worker Id	Address	State	Cores	Memory
worker-20190606231405-10.0.0.8-45591	10.0.0.8:45591	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20190606231405-10.0.0.9-38087	10.0.0.9:38087	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration

Completed Applications (1)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20190606233148-0000	csv-hdfs-mongo	2	1024.0 MB	2019/06/06 23:31:48	root	FINISHED	40 min

After scaling up the process was much faster: 13 min using 5 cores and 2GB of memory per worker

Spark Master at spark://master:7077

URL: spark://master:7077
Alive Workers: 5
Cores in use: 5 Total, 0 Used
Memory in use: 10.0 GB Total, 0.0 B Used
Applications: 0 Running, 1 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (5)

Worker Id	Address	State	Cores	Memory
worker-20190607130958-10.0.0.55-36811	10.0.0.55:36811	ALIVE	1 (0 Used)	2.0 GB (0.0 B Used)
worker-20190607130958-10.0.0.62-39269	10.0.0.62:39269	ALIVE	1 (0 Used)	2.0 GB (0.0 B Used)
worker-20190607131002-10.0.0.56-35305	10.0.0.56:35305	ALIVE	1 (0 Used)	2.0 GB (0.0 B Used)
worker-20190607131003-10.0.0.53-43671	10.0.0.53:43671	ALIVE	1 (0 Used)	2.0 GB (0.0 B Used)
worker-20190607131003-10.0.0.60-35327	10.0.0.60:35327	ALIVE	1 (0 Used)	2.0 GB (0.0 B Used)

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration

Completed Applications (1)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20190607135730-0000	csv-hdfs-mongo	5	1024.0 MB	2019/06/07 13:57:30	root	FINISHED	13 min

Load the data with Jupyter notebook from MongoDB

Use jupyter notebook to load the data from MongoDB and then run some queries.

Go to Jupyter at http://localhost:8888

Type in the password that was used as the environment variable, in this case "project".

Alternatively:

Grab the token by looking at the logs:

```
docker service logs spark_jupyter
```

Use the startup_default_mongo.ipynb notebook to get started. Notice that in a notebook you must specify the input and output uri and the spark connector.

```
.config('spark.jars.packages','org.mongodb.spark:mongo-spark-connector_2.11:2.4.0' ) \
.config("spark.mongodb.input.uri", "mongodb://" + login + "@mongo/power.data?authSource=admin") \
.config("spark.mongodb.output.uri", "mongodb://" + login + "@mongo/power.data?authSource=admin")
```

jupyter startup_default_mongo Last Checkpoint: a minute ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Markdown

Load and Save files from MongoDB

In [1]:
```
user = %env MONGO_INITDB_ROOT_USERNAME
password = %env MONGO_INITDB_ROOT_PASSWORD
login = user + ":" + password
```

In [2]:
```
from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession

spark = SparkSession \
    .builder \
    .appName("jupyter") \
    .master("local") \
    .config('spark.jars.packages','org.mongodb.spark:mongo-spark-connector_2.11:2.4.0' ) \
    .config("spark.mongodb.input.uri", "mongodb://" + login + "@mongo/power.data?authSource=admin") \
    .config("spark.mongodb.output.uri", "mongodb://" + login + "@mongo/power.data?authSource=admin") \
    .getOrCreate()
```

In [3]:
```
df = spark.read.format("com.mongodb.spark.sql.DefaultSource").load()
```

In [4]:
```
df.show()
```

```
+-------+-------------+-------------------+-------------------+---------+--------------------+--------+
|  Acorn|Acorn_grouped|           DateTime|KWH_per_hh_per_30min|    LCLid|                 _id|stdorToU|
+-------+-------------+-------------------+-------------------+---------+--------------------+--------+
|ACORN-U|      ACORN-U|2012-05-03 05:30:00|              0.114|MAC000023|[5cf85c0a29582e00...|     Std|
|ACORN-U|      ACORN-U|2012-05-03 06:00:00|              0.177|MAC000023|[5cf85c0a29582e00...|     Std|
|ACORN-U|      ACORN-U|2012-05-03 06:30:00|              0.165|MAC000023|[5cf85c0a29582e00...|     Std|
|ACORN-U|      ACORN-U|2012-05-03 07:00:00|              0.307|MAC000023|[5cf85c0a29582e00...|     Std|
|ACORN-U|      ACORN-U|2012-05-03 07:30:00|              0.547|MAC000023|[5cf85c0a29582e00...|     Std|
|ACORN-U|      ACORN-U|2012-05-03 08:00:00|              0.497|MAC000023|[5cf85c0a29582e00...|     Std|
|ACORN-U|      ACORN-U|2012-05-03 08:30:00|              0.657|MAC000023|[5cf85c0a29582e00...|     Std|
|ACORN-U|      ACORN-U|2012-05-03 09:00:00|               0.16|MAC000023|[5cf85c0a29582e00...|     Std|
|ACORN-U|      ACORN-U|2012-05-03 09:30:00|              0.107|MAC000023|[5cf85c0a29582e00...|     Std|
|ACORN-U|      ACORN-U|2012-05-03 10:00:00|              0.107|MAC000023|[5cf85c0a29582e00...|     Std|
|ACORN-U|      ACORN-U|2012-05-03 10:30:00|              0.106|MAC000023|[5cf85c0a29582e00...|     Std|
|ACORN-U|      ACORN-U|2012-05-03 11:00:00|              0.106|MAC000023|[5cf85c0a29582e00...|     Std|
|ACORN-U|      ACORN-U|2012-05-03 11:30:00|              0.168|MAC000023|[5cf85c0a29582e00...|     Std|
|ACORN-U|      ACORN-U|2012-05-03 12:00:00|              0.456|MAC000023|[5cf85c0a29582e00...|     Std|
|ACORN-U|      ACORN-U|2012-05-03 12:30:00|              0.442|MAC000023|[5cf85c0a29582e00...|     Std|
|ACORN-U|      ACORN-U|2012-05-03 13:00:00|              0.659|MAC000023|[5cf85c0a29582e00...|     Std|
|ACORN-U|      ACORN-U|2012-05-03 13:30:00|               0.62|MAC000023|[5cf85c0a29582e00...|     Std|
|ACORN-U|      ACORN-U|2012-05-03 14:00:00|              0.523|MAC000023|[5cf85c0a29582e00...|     Std|
|ACORN-U|      ACORN-U|2012-05-03 14:30:00|               1.15|MAC000023|[5cf85c0a29582e00...|     Std|
|ACORN-U|      ACORN-U|2012-05-03 15:00:00|              0.905|MAC000023|[5cf85c0a29582e00...|     Std|
+-------+-------------+-------------------+-------------------+---------+--------------------+--------+
only showing top 20 rows
```

Save results from Jupyter to MongoDB

Save to default output uri database and collection

```
df.write.format("com.mongodb.spark.sql.DefaultSource").mode("append").save()
```

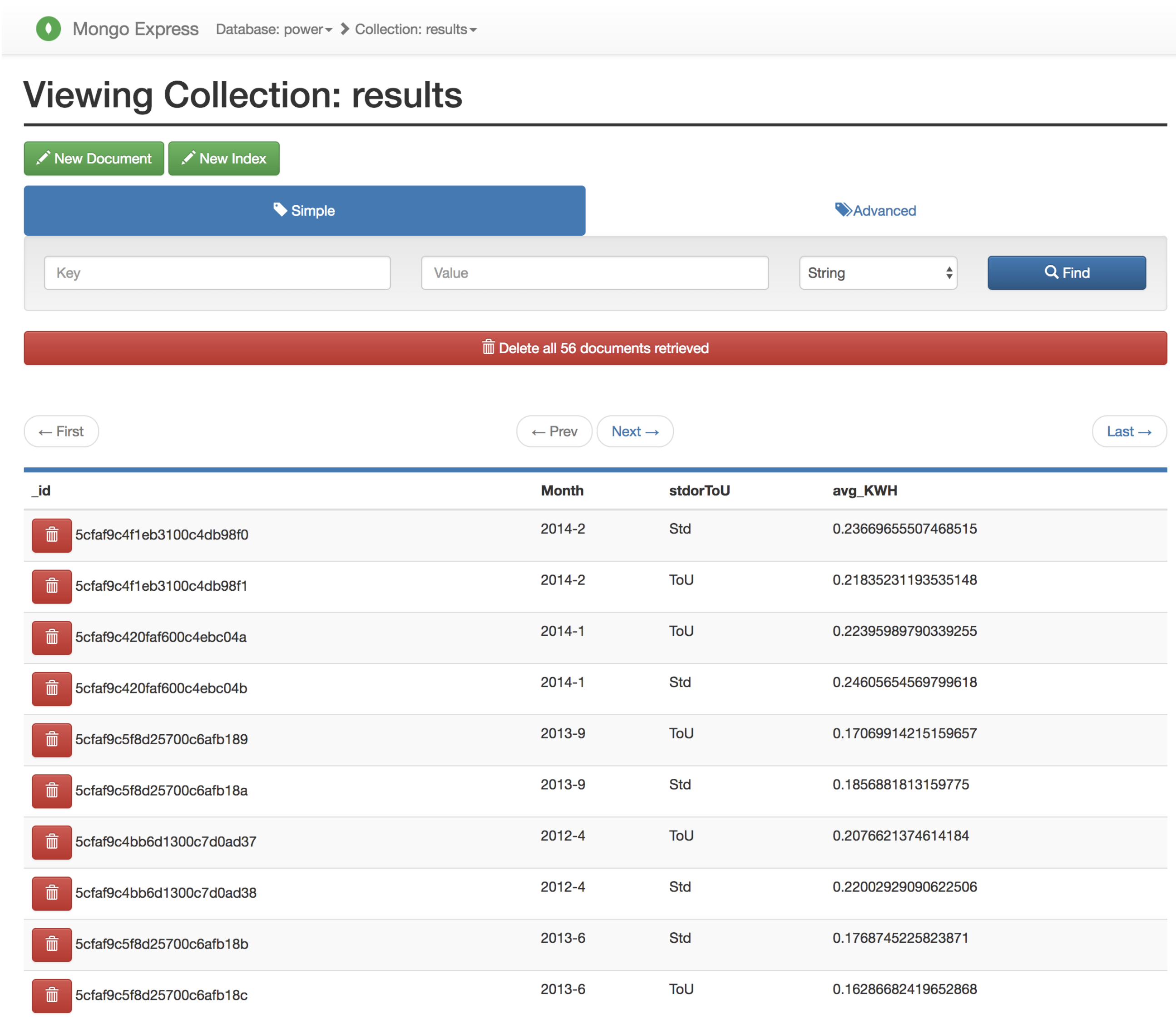
Save to a different db and coll

You must specify the output uri in the config before doing this. For a db:results and coll:data

```
.config("spark.mongodb.output.uri", "mongodb://" + login + "@mongo/results.data?authSource=admin")
```

Then write to that destination:

```
df.write.format("com.mongodb.spark.sql.DefaultSource").mode("append").option("database",
    "results").option("collection", "data").save()
```



_id	Month	stdorToU	avg_KWH
5cfaf9c4f1eb3100c4db98f0	2014-2	Std	0.23669655507468515
5cfaf9c4f1eb3100c4db98f1	2014-2	ToU	0.21835231193535148
5cfaf9c420faf600c4ebc04a	2014-1	ToU	0.22395989790339255
5cfaf9c420faf600c4ebc04b	2014-1	Std	0.24605654569799618
5cfaf9c5f8d25700c6afb189	2013-9	ToU	0.17069914215159657
5cfaf9c5f8d25700c6afb18a	2013-9	Std	0.1856881813159775
5cfaf9c4bb6d1300c7d0ad37	2012-4	ToU	0.2076621374614184
5cfaf9c4bb6d1300c7d0ad38	2012-4	Std	0.22002929090622506
5cfaf9c5f8d25700c6afb18b	2013-6	Std	0.1768745225823871
5cfaf9c5f8d25700c6afb18c	2013-6	ToU	0.16286682419652868

Security precautions with environment variables

Set environment variables in your local shell before deploying the stack.

```
export ME_CONFIG_MONGODB_ADMINUSERNAME=root
export ME_CONFIG_MONGODB_ADMINPASSWORD=example
export MONGO_INITDB_ROOT_USERNAME=root
export MONGO_INITDB_ROOT_PASSWORD=example
export JUPYTER_TOKEN='project'
```

Include these in the stack yml file. The values will be added to each service where they are specified

```
jupyter:
    image: jupyter/pyspark-notebook:f646d2b2a3af
```

```yaml
    volumes:
      - london-power:/home/jovyan/work
    ports:
      - 8888:8888
    environment:
      - MONGO_INITDB_ROOT_USERNAME
      - MONGO_INITDB_ROOT_PASSWORD
      - JUPYTER_TOKEN

  mongo:
    image: mongo
    restart: always
    environment:
      - MONGO_INITDB_ROOT_USERNAME
      - MONGO_INITDB_ROOT_PASSWORD

  mongo-express:
    image: mongo-express
    restart: always
    ports:
      - 8181:8081
    environment:
      - ME_CONFIG_MONGODB_ADMINUSERNAME
      - ME_CONFIG_MONGODB_ADMINPASSWORD
```

We used them to run the spark-submit in order to specify the uri. This allowed us to avoid using credentials in the py file itself

```
--conf "spark.mongodb.input.uri=mongodb://"$MONGO_INITDB_ROOT_USERNAME:$MONGO_INITDB_ROOT_PASSWORD"@mongo/power.data?
authSource=admin" \
--conf "spark.mongodb.output.uri=mongodb://"$MONGO_INITDB_ROOT_USERNAME:$MONGO_INITDB_ROOT_PASSWORD"@mongo/power.data?
authSource=admin" \
    /usr/spark-2.4.1/script/csv-hdfs-mongo.py
```

We then also use them in the jupyter notebook to avoid obviously exposing the credentials. They can still be accessed but they're not written directly into the notebook at least.

```python
user = %env MONGO_INITDB_ROOT_USERNAME
password = %env MONGO_INITDB_ROOT_PASSWORD
login = user + ":" + password


    .config("spark.mongodb.input.uri", "mongodb://" + login + "@mongo/power.data?authSource=admin") \
    .config("spark.mongodb.output.uri", "mongodb://" + login + "@mongo/power.data?authSource=admin") \
```

An alternative for the jupyter file would have been to set the uri in the spark stack so it wouldn't have to be referenced in a cell. This is like the spark-submit approach. Unfortunately, we weren't sure how to set the uri at runtime. We also would like to further investigate Docker Secrets.

The os.environ.get is also useful within python files. The magic %env that we used is better for notebooks.

```python
import os
user = os.environ.get('MONGO_USER')
password = os.environ.get('MONGO_PASS')
login = user + ":" + password
```

Analyze the data with Spark Dataframes

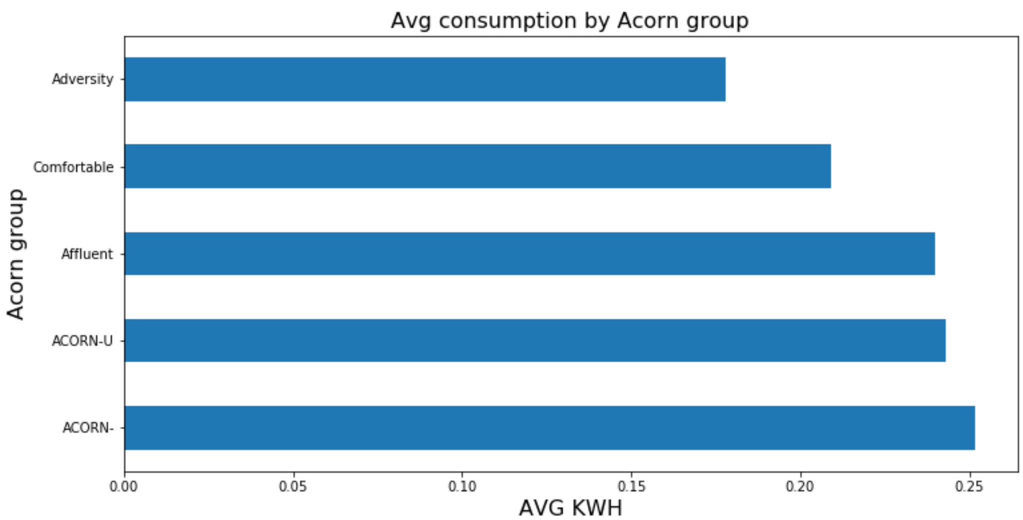
We used Spark to connect to our MongoDB with all the data that we uploaded. For the full 11GB file however, we found it faster to read from HDFS so we used MongoDB to save the results instead. We used both pythonic syntax and SparkSQL syntax to demonstrate.

See analysis_final.ipynb for the specifics.

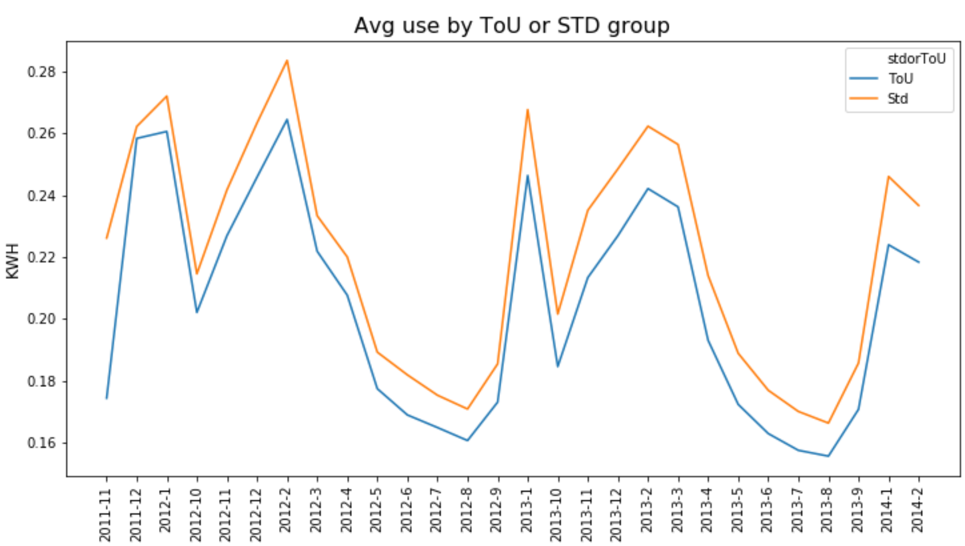
In summary, we read the file in from HDFS as a dataframe:

```
df = spark.read.parquet('hdfs://hadoop:8020/data/power')
```

We confirmed there were 167,926,914 records. We then set out to find the monthly consumption patterns of each Acorn group and plot them. On average, the energy consumption was highest in "affluent" households, followed by "comfortable" and lowest in "adversity" households.



Furthermore, we wanted to see if the test group that received warnings about energy prices beforehand consumed less energy than those who did not. The results showed that they did.



Finally, we saved the results to a new MongoDB collection:

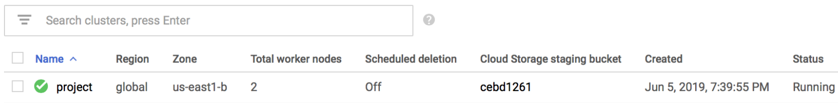
```
results.write.format("com.mongodb.spark.sql.DefaultSource").mode("append").option("database", \
"power").option("collection", "results").save()
```

Spark with Google Cloud Platform

Running a spark submit on GCP

Create project called cebd1261

Create a cluster called `project` with 2 workers



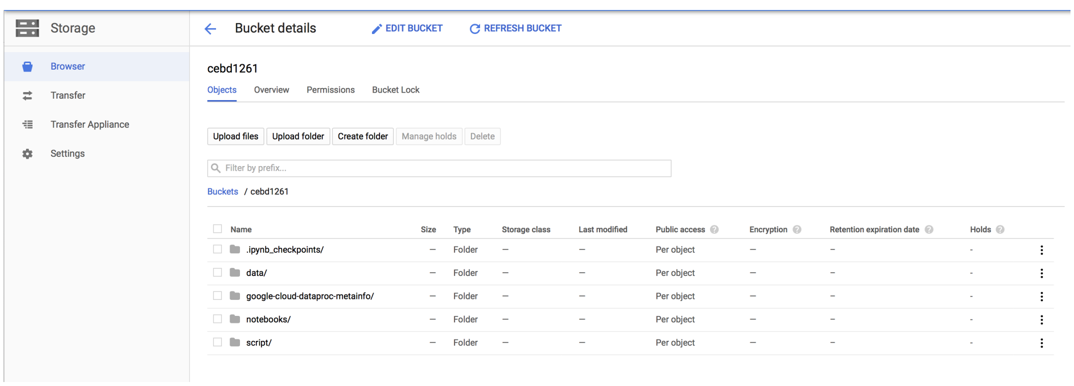
Create and populate the bucket with `/data` and `/script` using `Regional Storage` as `Storage Class` .This class is optimal for our need as it is for storing frequently accessed data in the same region as `Google CloudProc` or `Google Compute Engine` instances that use it such as data analytics and is lower cost per GB than Multi-Regional Storage used in class tutorial.

We add a label to our bucket for example 'project' so if we theoretically have many buckets they all are labeled for better access.

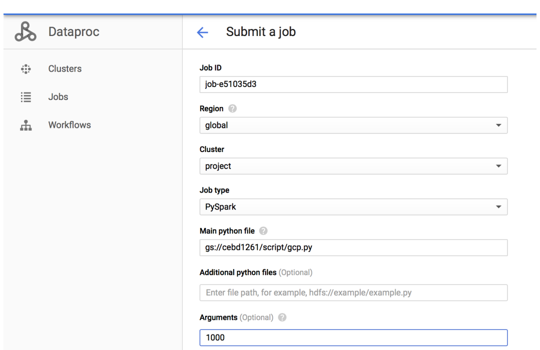
We aso can `Edit Permissions` to our bucket to add another user (like the other teammate for the project) so they can see what is inside inside the bucket (it can be read-only permission as well)

To work with storage through the command line we use `gsutil`

```
- `gsutil ls gs://cebd1261/` to see what is inside the bucket
- `gsutil ls -L` gives a lot information about the settings of the bucket
- `gsutil mb gs://cebd1261_project/`  to create a new bucket
- `gsutil mv gs://cebd1261/gcp.py gs://cebd1261_project/` to move a file from one bucket to another
- ...
```



Create and submit a spark job by choosing `PySpark` as `Job type` and pointing to our bucket for PyScript to run



Search jobs, press Enter

Job ID	Region	Type	Cluster	Start time	Elapsed time	Status
job-48f03a00	global	PySpark	jevfka-proj	Jun 5, 2019, 9:17:49 PM	28 sec	Succeeded

When Job output is complete we can see the results

job-48f03a00

Start time: Jun 5, 2019, 9:17:49 PM Elapsed time: 28 sec Status:

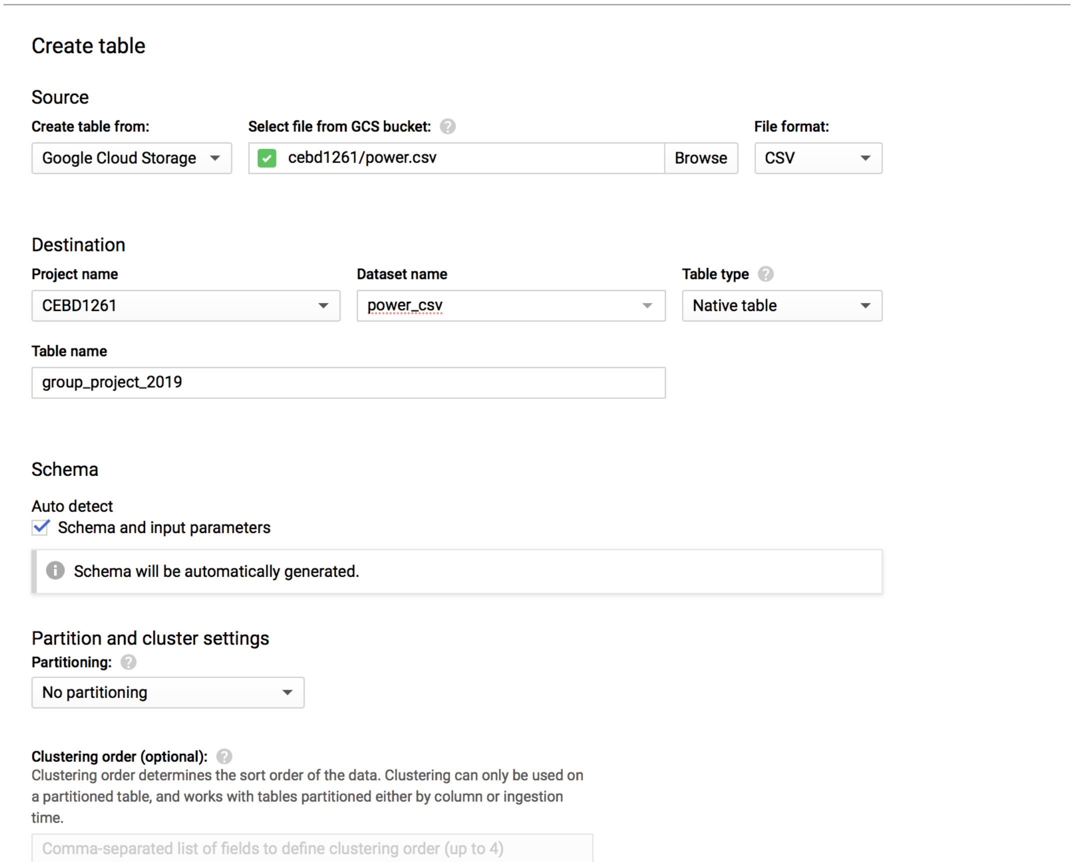
Output Configuration

Line wrapping Equivalent command

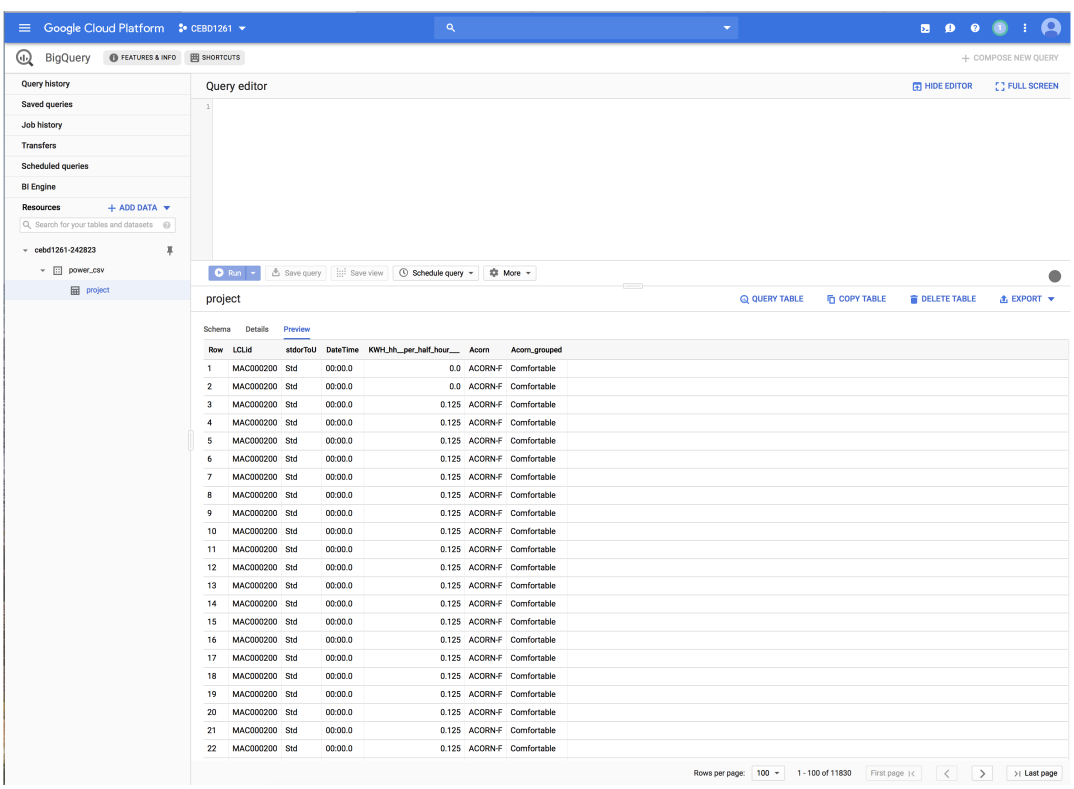
```
19/06/06 01:18:00 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted application application_1559783738287_0001
+---------+--------+-------------------+-------------------+-------+-------------+
|    LCLid|stdorToU|           DateTime|KWH_per_hh_per_30min|  Acorn|Acorn_grouped|
+---------+--------+-------------------+-------------------+-------+-------------+
|MAC000002|     Std|2012-10-12 00:30:00|                0.0|ACORN-A|     Affluent|
|MAC000002|     Std|2012-10-12 01:00:00|                0.0|ACORN-A|     Affluent|
|MAC000002|     Std|2012-10-12 01:30:00|                0.0|ACORN-A|     Affluent|
|MAC000002|     Std|2012-10-12 02:00:00|                0.0|ACORN-A|     Affluent|
|MAC000002|     Std|2012-10-12 02:30:00|                0.0|ACORN-A|     Affluent|
|MAC000002|     Std|2012-10-12 03:00:00|                0.0|ACORN-A|     Affluent|
|MAC000002|     Std|2012-10-12 03:30:00|                0.0|ACORN-A|     Affluent|
|MAC000002|     Std|2012-10-12 04:00:00|                0.0|ACORN-A|     Affluent|
|MAC000002|     Std|2012-10-12 04:30:00|                0.0|ACORN-A|     Affluent|
|MAC000002|     Std|2012-10-12 05:00:00|                0.0|ACORN-A|     Affluent|
|MAC000002|     Std|2012-10-12 05:30:00|                0.0|ACORN-A|     Affluent|
|MAC000002|     Std|2012-10-12 06:00:00|                0.0|ACORN-A|     Affluent|
|MAC000002|     Std|2012-10-12 06:30:00|                0.0|ACORN-A|     Affluent|
|MAC000002|     Std|2012-10-12 07:00:00|                0.0|ACORN-A|     Affluent|
|MAC000002|     Std|2012-10-12 07:30:00|                0.0|ACORN-A|     Affluent|
|MAC000002|     Std|2012-10-12 08:00:00|                0.0|ACORN-A|     Affluent|
|MAC000002|     Std|2012-10-12 08:30:00|                0.0|ACORN-A|     Affluent|
|MAC000002|     Std|2012-10-12 09:00:00|                0.0|ACORN-A|     Affluent|
|MAC000002|     Std|2012-10-12 09:30:00|                0.0|ACORN-A|     Affluent|
```

BigQuery

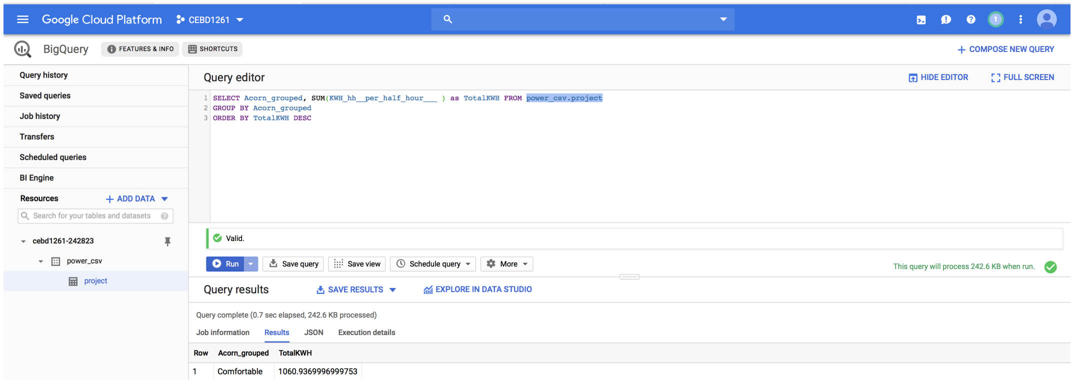
BigQuery on GCP can process petabytes in minutes. To query data we pin the project associated with data we are using and create a table. Then populate the table from the Google Storage bucket where our data is stored:



It is possible to load data from Google Cloud bucket as well as from a local machine and other sources. Adding data is incredibly easy. We had some trouble with `nulls`. To solve this in `Advanced settings` in table creating pane click `ignore unrecognized characters` the database was created and nulls ignored.



Running a query using a simple SQL language.The queries can be scheduled to run as needed. There are two types of pricing for running queries either flat rate or per query. After running each query the pane shows how many Kb the process took and that way it is easy to plan what pricing is better.

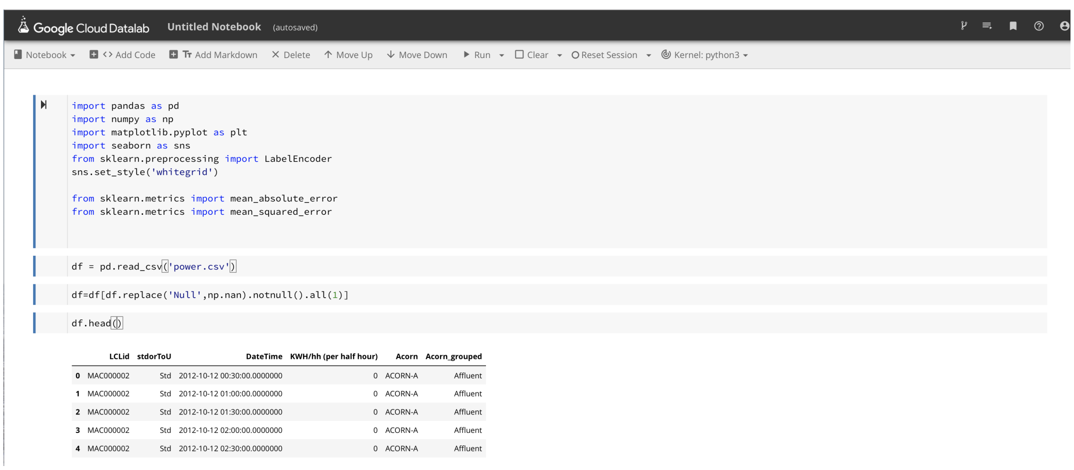


Jupyter notebooks in a Cloud Datalab Session

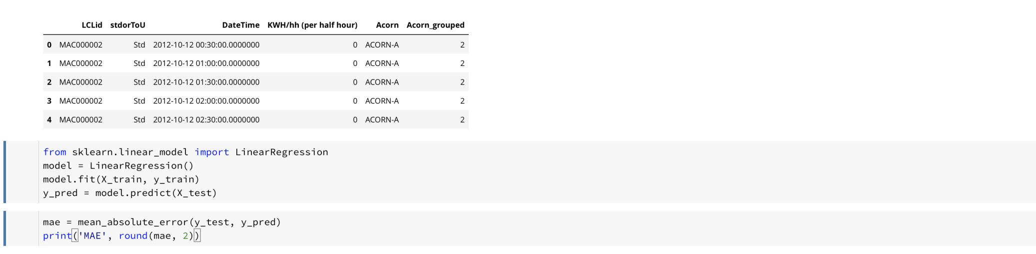
`Cloud Datalab` runs on `Compute Engine` and connects to many cloud services running on a vm instance. It has built-in `Jupyter Notebooks` that supports Tensorflow and SkLearn and can analyze terabytes of data hence often used for Machine learning.

First open secure shell and run `gcloud components install datalab` > then run `datalab create datalabvm` > choose zone (say us-east-b) > console will output the localhost port that `Datalab` is available at > choose `web preview` and change port accordingly if necessary

We run a simple Machine Learning task predicting Acorn_grouped depending on KWH per half an hour



We get a prediction Mean squared error of 0.39

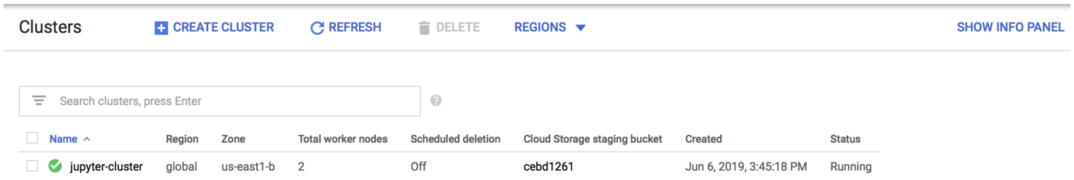


Install and run a Jupyter notebook on a Cloud Dataproc cluster

Create a cluster and install the Jupyter component through the shell

```
gcloud beta dataproc clusters create jupyter-cluster \
    --optional-components=ANACONDA,JUPYTER \
    --image-version=1.3 \
    --enable-component-gateway \
    --bucket cebd1261 \
    --project cebd1261-242823
```

Navigate to the `Cloud Dataproc Clusters` , then select our cluster to open the `Cluster details` form. Click the `Web Interfaces`



We can open a PySpark notebook stored in a bucket and work with data also stored inside our bucket by pointing to it during read:

```
spark.read.csv('gs://cebd1261/data/power.csv', schema=final_struc, header=True)
```

```
In [2]: from pyspark.context import SparkContext
        from pyspark.sql.session import SparkSession
        # sc = SparkContext('local')
        # spark = SparkSession(sc)

In [3]: from pyspark.sql.types import (StructField, StringType,
                                       IntegerType, TimestampType,
                                       FloatType, StructType)

In [4]: data_schema = [StructField('LCLid', StringType(), True),
                       StructField('stdorToU', StringType(), True),
                       StructField('DateTime', TimestampType(), True),
                       StructField('KWH_per_hh_per_30min', FloatType(), True),
                       StructField('Acorn', StringType(), True),
                       StructField('Acorn_grouped', StringType(), True)]

        final_struc = StructType(fields=data_schema)

In [5]: power = spark.read.csv('gs://cebd1261/data/power.csv', schema=final_struc, header=True)

In [6]: power.printSchema()

        root
         |-- LCLid: string (nullable = true)
         |-- stdorToU: string (nullable = true)
         |-- DateTime: timestamp (nullable = true)
         |-- KWH_per_hh_per_30min: float (nullable = true)
         |-- Acorn: string (nullable = true)
         |-- Acorn_grouped: string (nullable = true)

In [7]: power.count()

Out[7]: 1000000

In [8]: print('The total number of rows is:', power.count(), '\nThe total number of columns is:', len(power.columns))

        ('The total number of rows is:', 1000000, '\nThe total number of columns is:', 6)

In [9]: power.show(50)

        +---------+--------+-------------------+--------------------+-------+-------------+
        |    LCLid|stdorToU|           DateTime|KWH_per_hh_per_30min|  Acorn|Acorn_grouped|
        +---------+--------+-------------------+--------------------+-------+-------------+
        |MAC000002|     Std|2012-10-12 00:30:00|                 0.0|ACORN-A|     Affluent|
        |MAC000002|     Std|2012-10-12 01:00:00|                 0.0|ACORN-A|     Affluent|
        |MAC000002|     Std|2012-10-12 01:30:00|                 0.0|ACORN-A|     Affluent|
        |MAC000002|     Std|2012-10-12 02:00:00|                 0.0|ACORN-A|     Affluent|
        |MAC000002|     Std|2012-10-12 02:30:00|                 0.0|ACORN-A|     Affluent|
        |MAC000002|     Std|2012-10-12 03:00:00|                 0.0|ACORN-A|     Affluent|
        |MAC000002|     Std|2012-10-12 03:30:00|                 0.0|ACORN-A|     Affluent|
        |MAC000002|     Std|2012-10-12 04:00:00|                 0.0|ACORN-A|     Affluent|
        |MAC000002|     Std|2012-10-12 04:30:00|                 0.0|ACORN-A|     Affluent|
        |MAC000002|     Std|2012-10-12 05:00:00|                 0.0|ACORN-A|     Affluent|
        |MAC000002|     Std|2012-10-12 05:30:00|                 0.0|ACORN-A|     Affluent|
        |MAC000002|     Std|2012-10-12 06:00:00|                 0.0|ACORN-A|     Affluent|
        |MAC000002|     Std|2012-10-12 06:30:00|                 0.0|ACORN-A|     Affluent|
        |MAC000002|     Std|2012-10-12 07:00:00|                 0.0|ACORN-A|     Affluent|
        |MAC000002|     Std|2012-10-12 07:30:00|                 0.0|ACORN-A|     Affluent|
        |MAC000002|     Std|2012-10-12 08:00:00|                 0.0|ACORN-A|     Affluent|
        |MAC000002|     Std|2012-10-12 08:30:00|                 0.0|ACORN-A|     Affluent|
```

Some MLlib

In this Diploma we had opportunity to explore basics of SkLearn however we took this opportunity to learn a little bit about MLlib.During GCP part of this project we applied a quick Linear Regression model in sklearn to our data in DataLab trying to predict KHW/hh used based on household code and the Acorn_gropuded column that measures social status of the household. Ideally we wanted to use also the column where it indicates wheather the people were advised about the peaks in electricity beforehand,time of the day etc however the sample dataset had only one category and hence useless as a feature. When we tried to upload the 11G dataset to Hadoop the spark-submit would interrupt and the WARN would print that not enough resources were allocated(even after changing amountof spark_workers to 3 and memory to 2g in spark-submit .yml file)We also tried DecisionTreeRegressor, RandomForestRegressor, GBTRegressor with poor results.

Mean Squared error with SkLearn was 0.27 and with MLlib 0.20 which surprised us since we expected same results for both. In the future more models should be trained using more accurate feature engineering. Notebooks available in our repo.

Future Improvements

- Make multi-node cluster
- Fully automated with a scheduler like airflow
- Machine Learning with Spark MLlib
- Improve security
- Connect MongoDB to Dataproc

Conclusion

Through the project we refined our docker skills and explored various file types and storage options. We got to really see the power of distributed Spark jobs by running our script on many workers with an 11GB file. We also explored new territory on Google Cloud Platform, scratching the surface of running Spark in a cloud environment. We briefly explored the data with Spark dataframes. We look forward to using more advance Spark features like MLib.