# Assignment 4

Name – Sati, Ankit Date – 3/17/2022

Section – 001

SID – as14128

Total in points (Maximum 100 points)–

Professors Comments –

Affirmation of Independent Effort – Ankit Sati

**Index**

**Note to Grader** - We have discussed with the professor that we only have to complete **4 of the 7 topics** in the assignment.

**However I have done 6 and have only missed part 4.**

Have discussed the same with professor as **HDInsight is a paid service.**

**Part I - Completed**

**Part II - Completed**

**Part V - Completed**

**Part III - Completed**

**Part VI - Completed**

**Part VII - Completed**

**Part VII - HDinsight is a paid service – left this one. (Explained above)**

**Part I – Get Familiar with spark.**

1. **Setting up resources to run the Spark Euler.**

Text

Description automatically generated

1. **Setting up the Eular-Spark in the same directory.**

* **$ export PYSPARK\_DRIVER\_PYTHON=Jupyter**
* **$ export PYSPARK\_DRIVER\_PYTHON\_OPTS=notebook**

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

**Second Notebook just to experiment with. I choose SQL-Magic**

* pip install pandas
* pip install sqlalchemy # ORM for databases

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

* pip install ipython-sql # SQL magic function

Text

Description automatically generated

Text

Description automatically generated

1. **Sql-magic**

Finally now we can go ahead and setup the SQL magic over the Jupyter Notebook.

We will be using the same notebook as shared in the example in the tutorial.

Text

Description automatically generated

Table

Description automatically generated

Now finally we need to check the build of this data on the SQL notebook, for that we will just collect the data provided in the first shard.

Total number of records

* We will have have a total of 50 records in the entire file

Command – sqlContect.sql(‘A,B,C’)

Graphical user interface, text, application

Description automatically generated

**Part II: Experiment with Spark on AWS.**

**Amazon Elastic Map Reduce (EMR)**

* EC2(**Compute server**) – This is basically a **regular server** instance that is used to deploy and the required resources over the VM as per the choices made by the users.

- This is used to deploy the VM.

- Manage resources over those VM’s.

- Finally to migrate services and monitor volumes.

* S3(**Storage utility**) – This is a basic protocol that acts like a storage bucket.

- The prime feature of this protocol is to deal with the data as per service request.

- We need this to store the data in the **data buckets** which are later used to store and move the data across volumes created.

No we need to setup the cluster and run a stream to analyze on top of it.

Graphical user interface, text, application

Description automatically generated

Experimenting with Elastic Map Reduce on the minimal cluster.

**Step 2 –**Graphical user interface, text, application, email

Description automatically generated

Amazon Cluster Info

Graphical user interface, text, application

Description automatically generated

**We need to make sure that the stream is ready.**

Graphical user interface, text, application, email

Description automatically generated

Next we need to begun the EMR and work with the yarn data in livy2.

IP address - **“hdfs://ip-172-31-22-51.ec2.internal:8020/user/wiki/text.txt”**

Text file = **txtfile.repartion(10)**

Array - Finally we need to put these values onto the array in ses2

Graphical user interface, text, application, email

Description automatically generated

Text file = **txtfile.repartion(10)**

Splitting this test lines into black spaces.

Graphical user interface, text, application, email

Description automatically generated

**Now finally we will complete the experiment by extracting the heading from each and every line.**

Graphical user interface, text, application

Description automatically generated with medium confidence

**Part V: Experiment with Streaming Big Data on AWS; execute and document the following Exercise.**

Note – In this case I have used the same json repository mentioned in tutorial as if we take the other one it will shoot up the cost because we have chosen provisioned data.

File chosen - .json available with tutorial.

1. Setup the kinesis account on amazon and make a new strem. Stream name – as14128-stream. This is just a dummy stream without any data as of this moment.

Graphical user interface, text, application, email

Description automatically generated

1. Next step is to check if all the things are in place and see that the monitoring **charts** are empty.

Graphical user interface, text, application, email

Description automatically generated

1. Now that we have checked that the charts are empty, we need to go ahead make sure that we have noted don the **configuration** for this stream as it will come in handy later in the process.

* Capacity Mode – Provisioned
* Total Shards -1 (Only passing a single file onto this stream so no point in getting more)
* Write capacity on Stream – 1Mib/Second (The data will come in pretty quick so need to stop to reduce the cost)
* Read capacity – 2 Mib/second

A screenshot of a computer

Description automatically generated

Text

Description automatically generated

1. Now we need to create a new Stack on top op which we are going to overlay the data. In our case we are taking a small .json file so we will be using the basic stack.

Creating account on KDG

Graphical user interface, text, application, email

Description automatically generated

A picture containing text

Description automatically generated

1. Now we need to start creating the stack on this profile and monitor for events to be created.

Graphical user interface, application

Description automatically generated

1. Now we need to start passing the .json file onto the stack that we had created on our KDG.

No we need to be quick because as soon as the data is released the entire burst will come onto the stack and we will need to stop it quick.

Graphical user interface, application

Description automatically generated

Passing a Valid input depending upon the test file in exercise.

Graphical user interface, text, application

Description automatically generated

Tracking the progress to monitor the charts later.

Graphical user interface, text, application

Description automatically generated

1. Now we need to be quick as time is money. The burst has begin and we will soon see the influx of data in the empty charts below.

Graphical user interface, text, application, email

Description automatically generated

Just to check the influx of data we will need to one of the chart to see if the data overlays properly onto this stack.

Chart

Description automatically generated with medium confidence

Chart, line chart

Description automatically generated

Since this method cost money for as long as it stays up, I have taken it down within a minute post recording the **completion**  of the activity.

1. Finally I will be deleting the resources from KDG first.

A screenshot of a computer

Description automatically generated

1. Then finally, I will delete the main Kinesis Datastream as well to cut down any further resource utilization.

Graphical user interface, text, application, email

Description automatically generated

**Part III: Google Datalab**

1. We need to setup both of the notebooks on the google labs CLI.
2. Notebook 1

Text

Description automatically generated

1. Checking both the notebooks in the system GUI.

Graphical user interface, text, email

Description automatically generated

1. Check the basic lab information required for this experiment.

Graphical user interface, text, application, email

Description automatically generated

1. Finally uploading both the datalab notebooks to the google cloud.

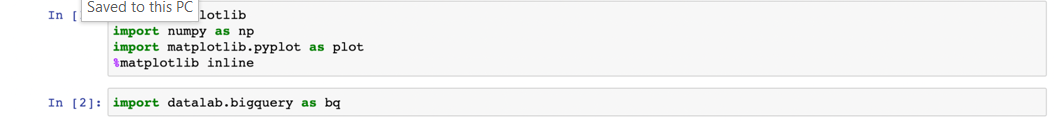
Background pattern

Description automatically generated

* **Datalab I**

In this first lab we are trying to track the rate of spread between the two cities that are separated over a distance of 2000 miles.

Having said that we need to run this on a single notepook.



Checking the same inputs in the reports before commit.

Graphical user interface, application

Description automatically generated

Now we need to track the activity on the graph by importing the “**rcParams**” from the same pyLab.

Chart, line chart

Description automatically generated

Graphical user interface, chart, line chart

Description automatically generated

Next we need to compare the two years. The data stored in lab 1 , will go through the same normalization and plot the boundaries for the next 4 years.

* **Datalab II –** Abnormality in the weather stations in Washington.

Graphical user interface, text, application, email

Description automatically generated

By tracing the below graph, we can see that the skagit station is the culprit.

Apart from this we can see the anomalies in the month of September and March for the same year.

Chart, line chart, histogram

Description automatically generated

**Part VI: Google Datalab**

This experiment was done on an EMR Hadoop cluster as well. A program was executed as a spark batch processing job to count the sum of the first 1 million prime numbers

Python code used is mentioned below.

Text

Description automatically generated

Final build and answer of the above .py file.

Text

Description automatically generated

**Part VII: Build additional experiments to document the use of MapReduce using Hadoop.**

1. Create a generic mapper function that calculates the number of words in the text file put in hdfs.

Text

Description automatically generated

1. Now we need to put the reducer file to count the number of words in the document.

Text

Description automatically generated

1. Submit both the application and monitor the status of map reduce over the files.

Text

Description automatically generated

1. Checking the count of a single sentence.

Text

Description automatically generated

1. Matching results of the book. (Initial file)

Text

Description automatically generated