Lab 3: EMR Lab

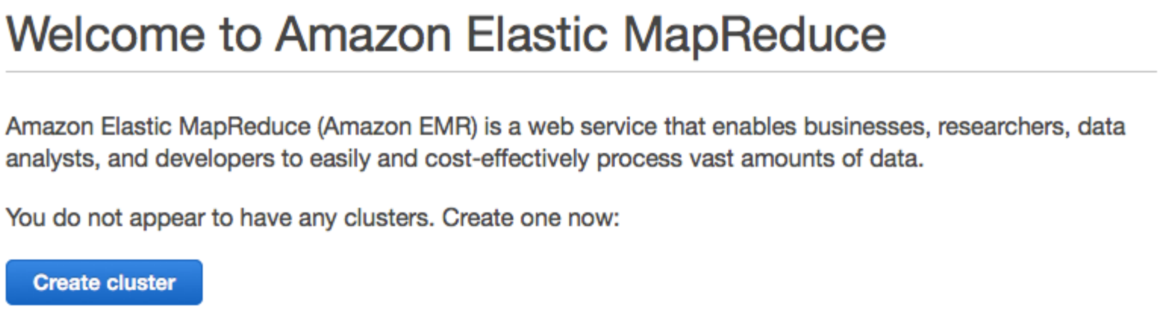
Section 1: Use Hive to query the AWS Data Catalog

Please ensure that you use **N. Virginia (us-east-1)** region for this lab.

Launch EMR Cluster

To launch a cluster using the console (make sure you have key pair generated in this region, you will use later)

1. Choose **N. Virginia (us-east-1) Region**
2. Open the **Amazon EMR** console
3. Click **Create cluster**.



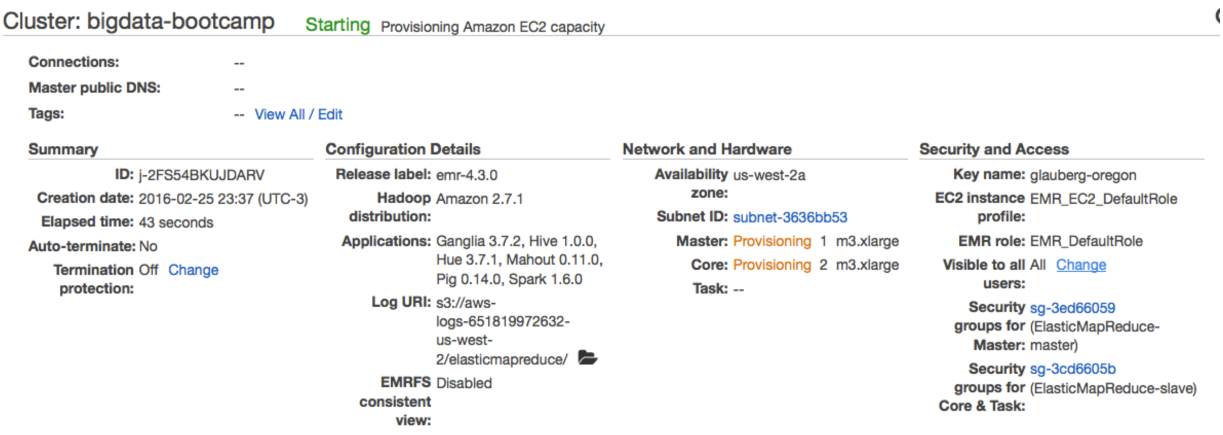
Welcome to Amazon Elastic MapReduce

Amazon Elastic MapReduce (Amazon EMR) is a web service that enables businesses, researchers, data analysts, and developers to easily and cost-effectively process vast amounts of data.

You do not appear to have any clusters. Create one now:

4. “Go to **“advance options”** and select **latest version**. Select “**Hadoop, Hive, Zeppelin, Spark, Hue, Ganglia,JupyterHub”**
5. AWS Glue Data Catalog settings:
    1. Select **Use for Hive table metadata**
    2. Select **Use for Spark table metadata**
    3.
6. Click **Next**
7. Instance Group Configuration -> **Uniform instance groups**
8. Network -> **Default VPC**
9. EC2 Subnet -> Choose any of the available subnets
10. Root device EBS: **10GB**
11. For for the machine types chose the following:
    1. Master node - 1 instance - m3.xlarge - On-Demand
    2. Core nodes - 1 instances – m3.xlarge - On-Demand
    3. Task nodes - 1 instances – m3.xlarge - Spot
12. Cluster name: **<user>_EMR_Cluster**
13. Logging: **Leave the bucket suggested**
14. Debugging: **On**
15. Termination protection: **On**
16. Scale down behavior: **Terminate at instance hour**
17. **Tags ->** Add a tag: **Name: user1**
18. Click **Next**
19. EC2 key pair: Choose **<user>** key pair

20. Permissions: **leave defaults**
21. Click **Create Cluster**
22. When the cluster starts, the console displays the **Cluster Details** page.



Cluster: bigdata-bootcamp Starting Provisioning Amazon EC2 capacity

Connections: --
Master public DNS: --
Tags: -- View All / Edit

Summary
ID: j-2FS54BKUJDARV
Creation date: 2016-02-25 23:37 (UTC-3)
Elapsed time: 43 seconds
Auto-terminate: No
Termination protection: Off Change

Configuration Details
Release label: emr-4.3.0
Hadoop distribution: Amazon 2.7.1
Applications: Ganglia 3.7.2, Hive 1.0.0, Hue 3.7.1, Mahout 0.11.0, Pig 0.14.0, Spark 1.6.0
Log URI: s3://aws-logs-651819972632-us-west-2/elasticmapreduce/
EMRFS consistent view: Disabled

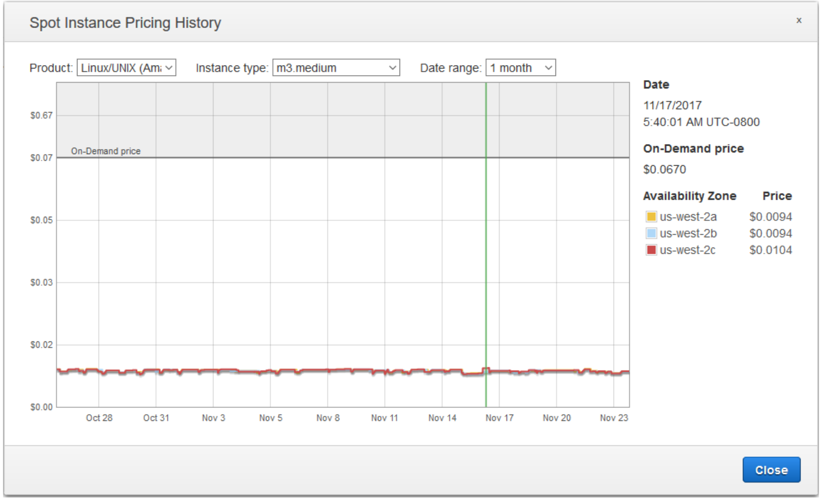
Network and Hardware
Availability zone: us-west-2a
Subnet ID: subnet-3636bb53
Master: Provisioning 1 m3.xlarge
Core: Provisioning 2 m3.xlarge
Task: --

Security and Access
Key name: glauberg-oregon
EC2 instance profile: EMR_EC2_DefaultRole
EMR role: EMR_DefaultRole
Visible to all users: All Change
Security groups for Master: sg-3ed66059 (ElasticMapReduce-master)
Security groups for Core & Task: sg-3cd6605b (ElasticMapReduce-slave)

It might take ~10 min for EMR to get to **Cluster ready** state.

While you are waiting take a look to price history of spot instances on the console:

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. On the navigation pane, choose **Spot Requests**.
3. If you are new to Spot Instances, you see a welcome page; choose **Get started**, scroll to the bottom of the screen, and then choose **Cancel**.
4. Choose **Pricing History**. By default, the page displays a graph of the data for Linux t1.micro instances in all Availability Zones over the past day. Move your pointer over the graph to display the prices at specific times in the table below the graph.
5. To review the Spot price history for a specific Availability Zone, select an Availability Zone from the list. You can also select a different product, instance type, or date range.



Spot Instance Pricing History

Product: Linux/UNIX (Ama
Instance type: m3.medium
Date range: 1 month

Date
11/17/2017
5:40:01 AM UTC-0800

On-Demand price
$0.0670

Availability Zone	Price
🟨 us-west-2a	$0.0094
🟦 us-west-2b	$0.0094
🟥 us-west-2c	$0.0104

Once the cluster is **Waiting** status, it's ready to process requests:

1. SSH to the Master node:

```
$ ssh -i <user>.pem hadoop@<Master_Node_DNS>
```

2. Open Hive:

```
$ hive
```

3. Query the **csv_streams** and **parquet_parquet** tables from the AWS Data Catalog:

```
hive> SELECT * from <user>.csv_stream LIMIT 10;
hive> SELECT * from <user>.parquet_parquet LIMIT 10;
```

Query also using spark-sql, type:

```
$ spark-sql
spark-sql>select * from <user>.csv_streams LIMIT 10;
```

Note: Please make sure that database and table name of the table is the correct one.

Section 2: Data Processing LAB - Analyzing Kinesis data with Amazon Spark on EMR

Overview.

In this exercise, you will learn how to process data using Spark running on EMR.

In the previous labs, we simulated a stream of events corresponding to some score, app ID, user ID and some app data that were streamed into a Kinesis stream. We processed them in real time using Kinesis Analytics, which is a service which allows to process streaming data in real time with standard SQL.

In this lab, we will use Spark running on EMR to implement a basic script that query data in our data lake.

Consumer Application

We are going to use Spark via Zeppelin: a web application that can execute code and represent any output table in multiple ways. To have access to Zeppelin application inside our EMR cluster, we need to enable web connection trough a SSH tunnel.

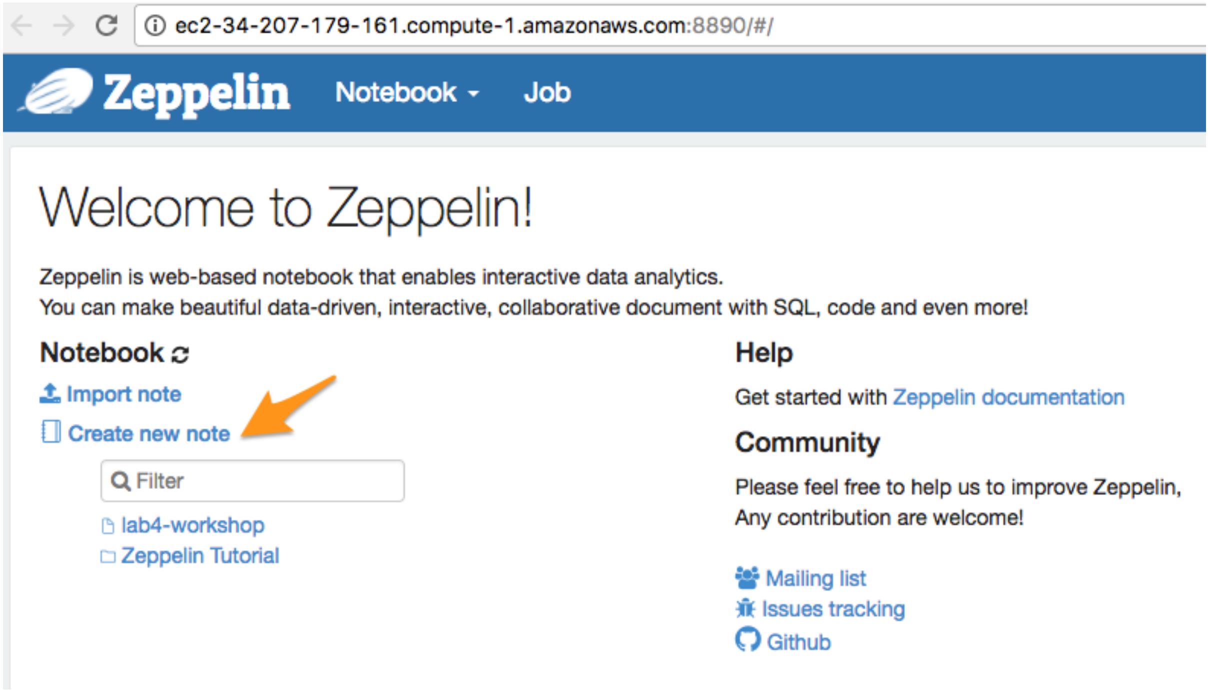
Enable Web Connection

1. Click on **Clusters** in the left menu.
2. Clink on your newly created EMR cluster: **`<user>_EMR_Cluster`**
3. Click on the tab: **Summary**
4. Set up a tunnel to port **8890** to be able to connect to the **Zeppelin Notebook** in the Master node of the EMR cluster:

```
$ ssh -i <user>.pem -L 8890:127.0.0.1:8890 hadoop@<MasterDNS>
```

5. You can open the **Zeppelin notebook** by typing this URL in the browser:

http://localhost:8890



Welcome to Zeppelin!

Zeppelin is web-based notebook that enables interactive data analytics.
You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!

Notebook

Import note

Create new note

Filter

lab4-workshop

Zeppelin Tutorial

Help

Get started with Zeppelin documentation

Community

Please feel free to help us to improve Zeppelin,
Any contribution are welcome!

Mailing list

Issues tracking

Github

6. Once you have logged in **Zeppelin**, click on: **Create new note**
7. Name the note: **SparkKinesis**
8. Choose interpreter: **Spark**
9. Click on: **Create note**

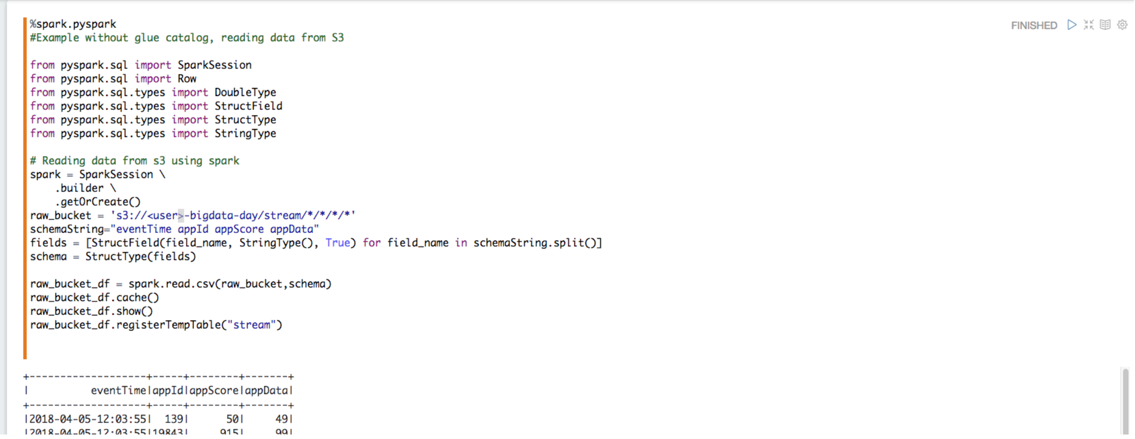
Run the following scripts below in the Spark Shell window to start using glue catalog via spark:

```
%spark.pyspark

#Example Using glue catalog via spark

spark.sql("show databases").show()

df = spark.sql("select * from <user>.csv_streams")

df.show()
```

```
%spark.pyspark
#Example Using glue catalog via spark

spark.sql("show databases").show()
df = spark.sql("select * from <user>.csv_streams")
df.show()
```

Run the following scripts below in the Spark Shell window to start analyzing your data without using glue catalog:

```
%spark.pyspark

#Example without glue catalog, reading data from S3

from pyspark.sql import SparkSession

from pyspark.sql import Row

from pyspark.sql.types import DoubleType

from pyspark.sql.types import StructField

from pyspark.sql.types import StructType

from pyspark.sql.types import StringType


# Reading data from s3 using spark

spark = SparkSession \

    .builder \

    .getOrCreate()

raw_bucket = 's3://<user>-bigdata-day/stream/*/*/*/*'

schemaString="eventTime appId appScore appData"

fields = [StructField(field_name, StringType(), True) for field_name in schemaString.split()]

schema = StructType(fields)


raw_bucket_df = spark.read.csv(raw_bucket,schema)

raw_bucket_df.cache()

raw_bucket_df.show()

raw_bucket_df.registerTempTable("stream")
```



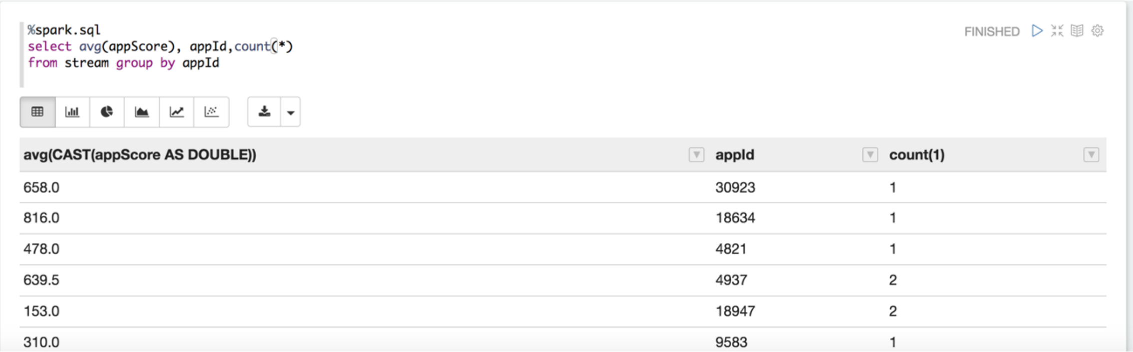
```
%spark.pyspark
#Example without glue catalog, reading data from S3

from pyspark.sql import SparkSession
from pyspark.sql import Row
from pyspark.sql.types import DoubleType
from pyspark.sql.types import StructField
from pyspark.sql.types import StructType
from pyspark.sql.types import StringType

# Reading data from s3 using spark
spark = SparkSession \
    .builder \
    .getOrCreate()
raw_bucket = 's3://<user>-bigdata-day/stream/*/*/*/*'
schemaString="eventTime appId appScore appData"
fields = [StructField(field_name, StringType(), True) for field_name in schemaString.split()]
schema = StructType(fields)

raw_bucket_df = spark.read.csv(raw_bucket,schema)
raw_bucket_df.cache()
raw_bucket_df.show()
raw_bucket_df.registerTempTable("stream")
```

```
+-------------------+-----+--------+-------+
|          eventTime|appId|appScore|appData|
+-------------------+-----+--------+-------+
|2018-04-05-12:03:55|  139|      50|     49|
|2018-04-05-12:03:55|19843|     915|     99|
```

NOTE: partitons keys are not included by default in spark unless path have format 'partitionkey=value' for example: s3://myBucket/yyyy=2018/mm=02/dd=01

Explore Spark SQL by typing **%sql** then Spark SQL query from the new line and click execute:

```
%spark.sql

select avg(appScore), appId,count(*)

from stream group by appId
```

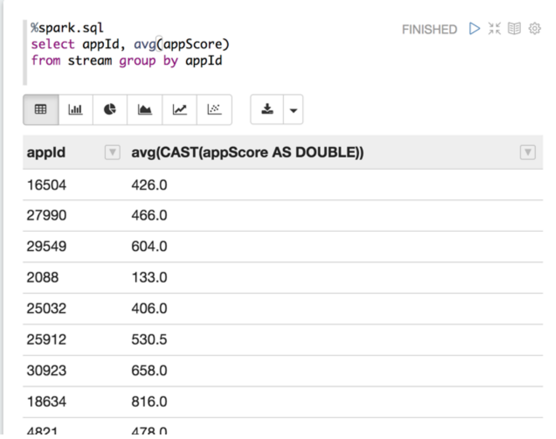


```
%spark.sql
select avg(appScore), appId,count(*)
from stream group by appId
```

avg(CAST(appScore AS DOUBLE))	appId	count(1)
658.0	30923	1
816.0	18634	1
478.0	4821	1
639.5	4937	2
153.0	18947	2
310.0	9583	1

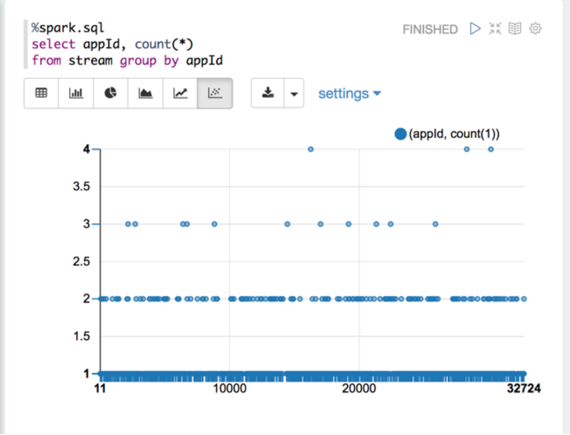
Adjust editor by using options icons on the right-hand side and Explore different charts with the following script.

```
%sql

select appId, avg(appScore) as avg_score from stream  group by appId
```



```
%spark.sql
select appId, avg(appScore)
from stream group by appId
```

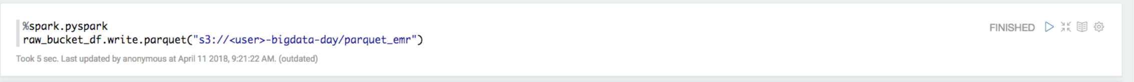
appId	avg(CAST(appScore AS DOUBLE))
16504	426.0
27990	466.0
29549	604.0
2088	133.0
25032	406.0
25912	530.5
30923	658.0
18634	816.0
4821	478.0



```
%spark.sql
select appId, count(*)
from stream group by appId
```

Run follow command in order to save as parquet format:

```
%spark.pyspark
raw_bucket_df.write.parquet("s3://<user>-bigdata-day/parquet_emr")
```



```
%spark.pyspark
raw_bucket_df.write.parquet("s3://<user>-bigdata-day/parquet_emr")
```

Took 5 sec. Last updated by anonymous at April 11 2018, 9:21:22 AM. (outdated)