PROJECT REPORT

2017

1. SHARED MEMORY SORT - JAVA

* Description of The Problem and Methodology

External sorting is a sorting algorithms that can handle massive amounts of data. External sorting is required when the data being sorted do not fit into the main memory of a computing device (usually RAM) and they must reside in the slower external memory (usually a hard drive).

External sorting typically uses a hybrid sort-merge strategy. In the sorting phase, chunks of data small enough to fit in main memory are read, sorted, and written out to a temporary file. In the merge phase, the sorted sub-files are combined into a single larger output file.

External sorting is the external merge sort algorithm, which sorts chunks that each fit in RAM, then merges the sorted chunks together. We first divide the file into **runs** such that the size of a run is small enough to fit into main memory. And these lists of strings are sorted in Sorted Linked List and written into temporary files. Finally merge the resulting runs together into successively bigger runs, until the file is sorted.

* Runtime Environment Settings and Details of the Instance

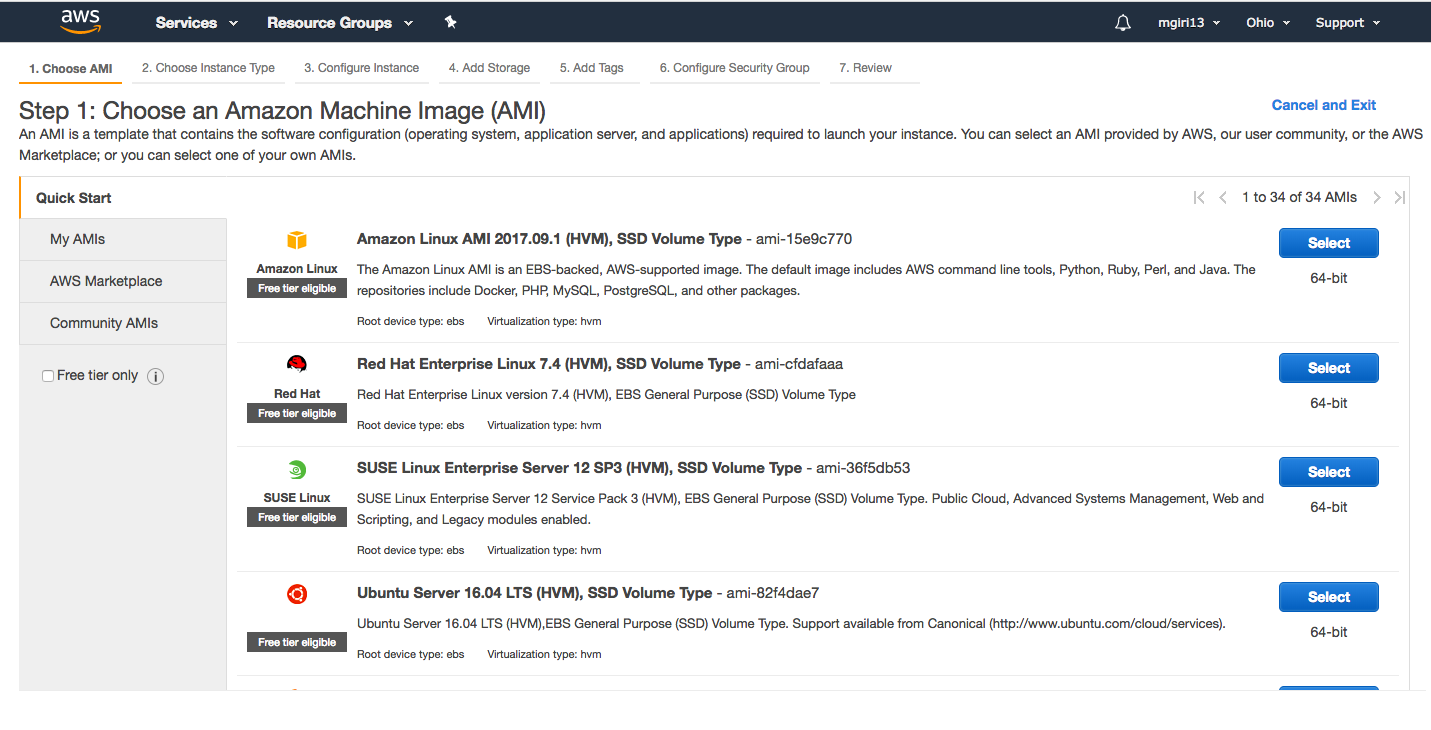
1. Host OS: Ubuntu Server 16.04LTS(HVM), SSD Volume Type
2. VERSION: JAVA 1.7
3. INSTANCE TYPE: i3. large
4. STORAGE: 420GB

Sort was performed in multi-threaded environment on 1 virtual node. Time to execute the Sort application on the 128GB and 1TB dataset produced with the Gensort on 1 node was calculated.

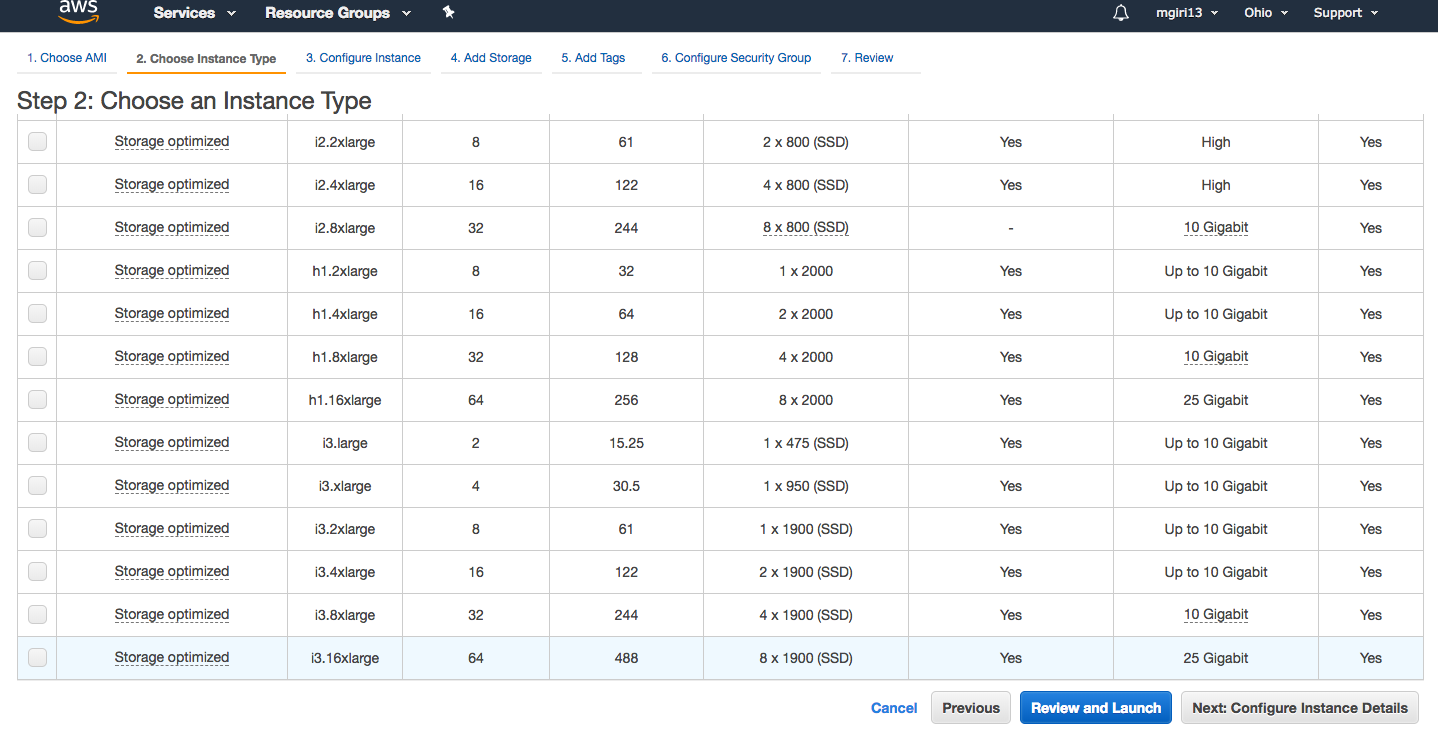
* Installation Steps to Setup Your Virtual Cluster and Difficulties Faced

I dint face any issues while setting up the instance. The steps are below:

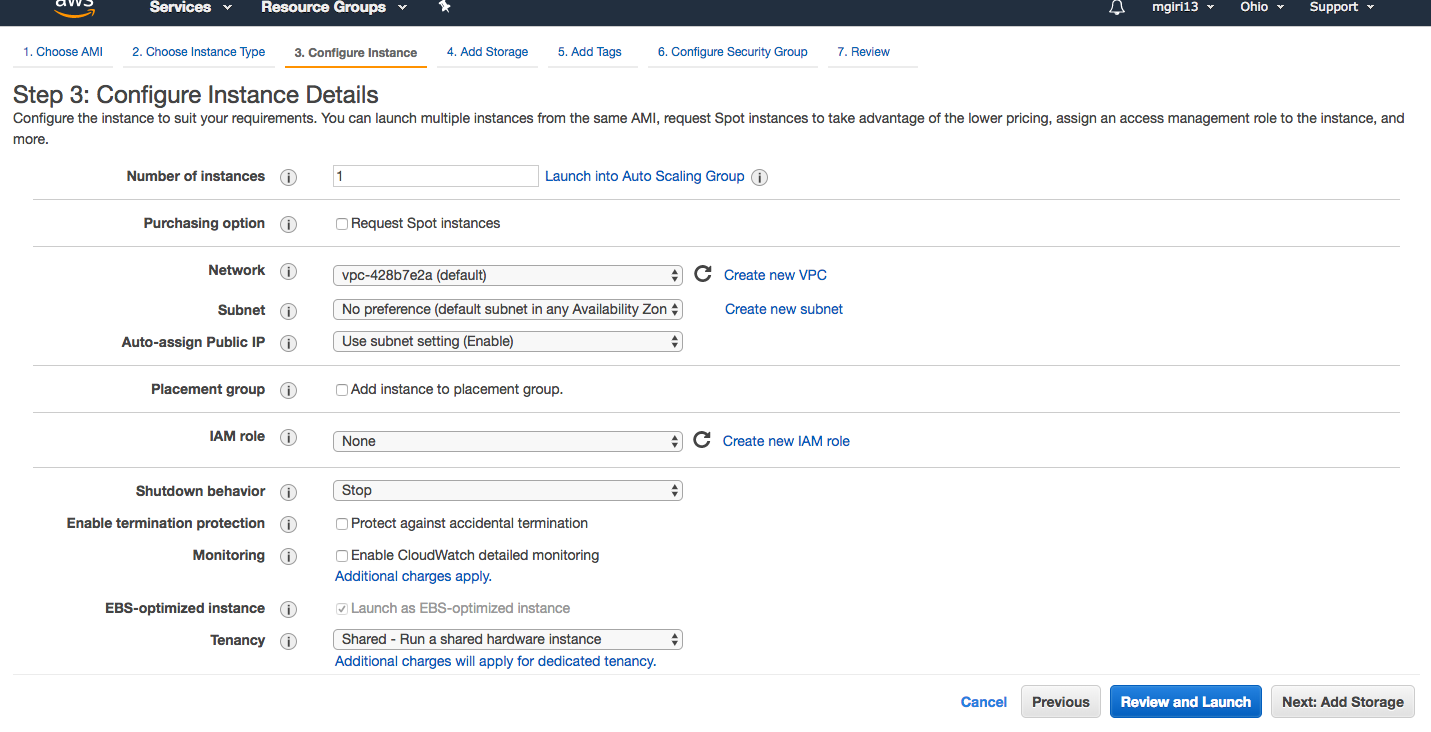
1. Select Ubuntu server and click next.



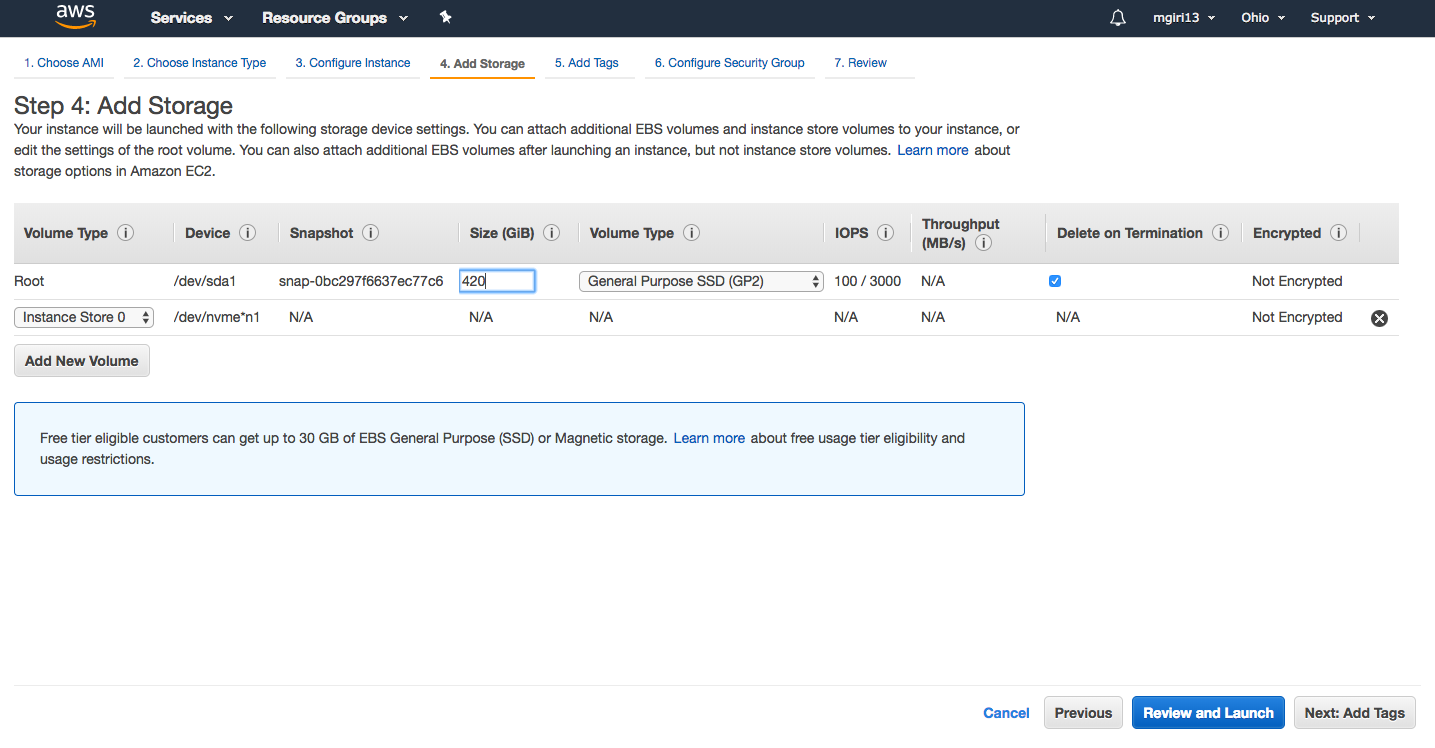
1. Choose the instance. Its i3.large for configuration1 and i3.4x.large for configuration 2. Click next.



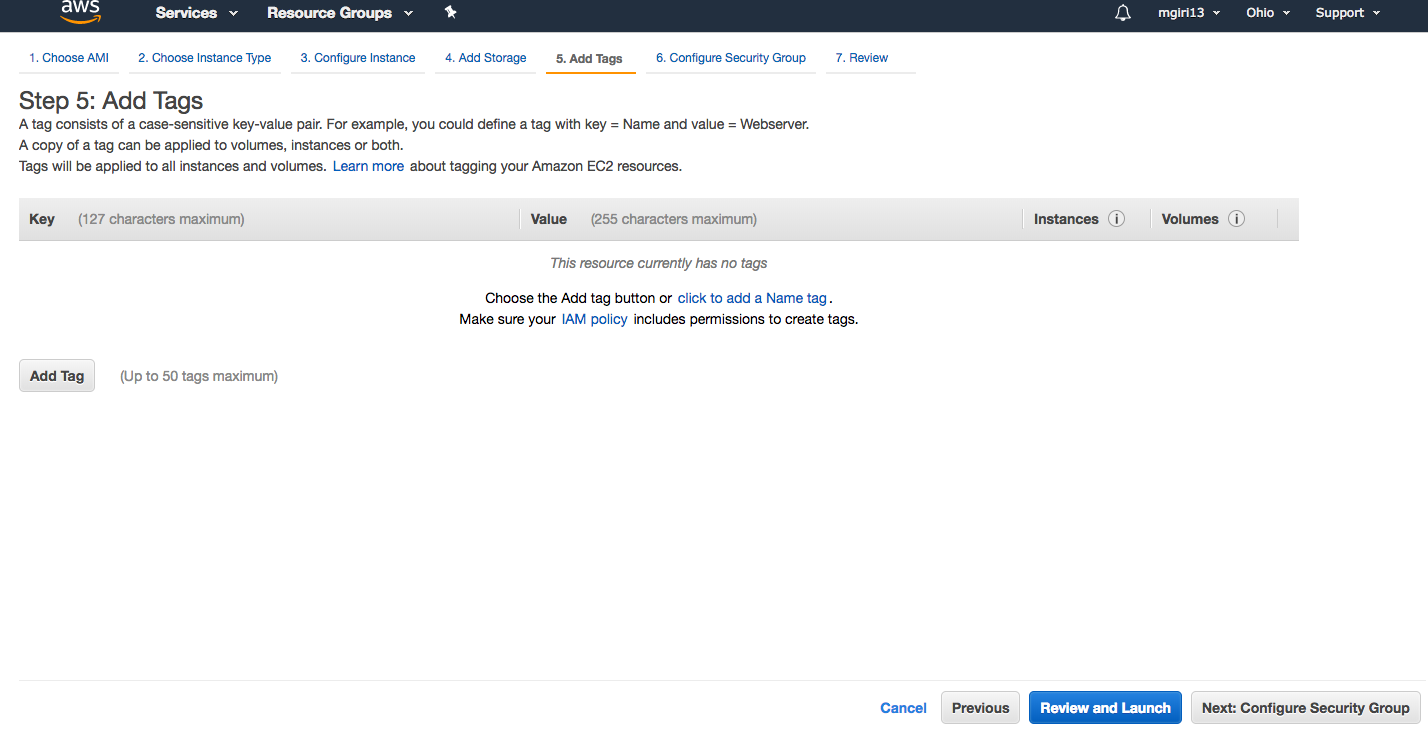
3. Check all the configuration and click next.



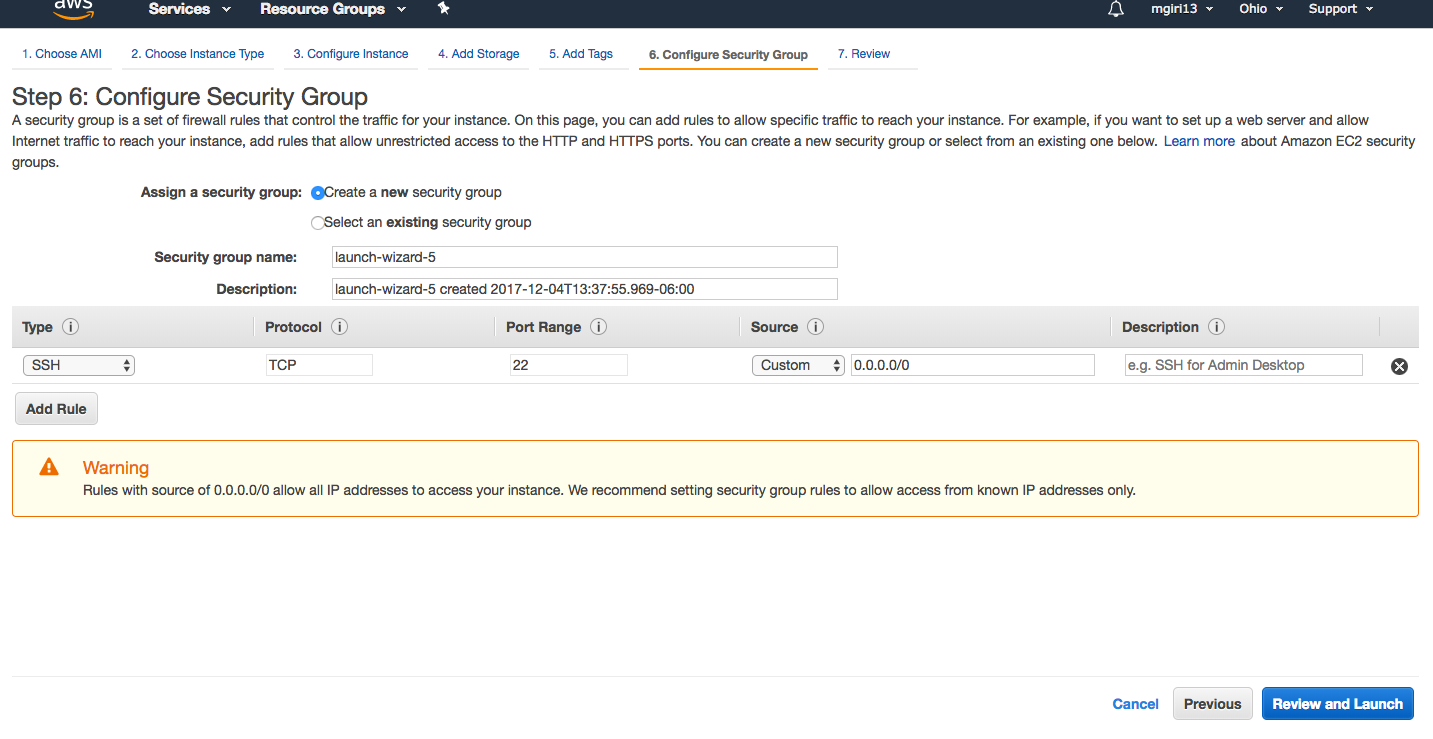
1. 4. Add the storage . For i3.large for configuration 1 add 430 GB and i3.4x.large for configuration2 add 3000GB . Click next.



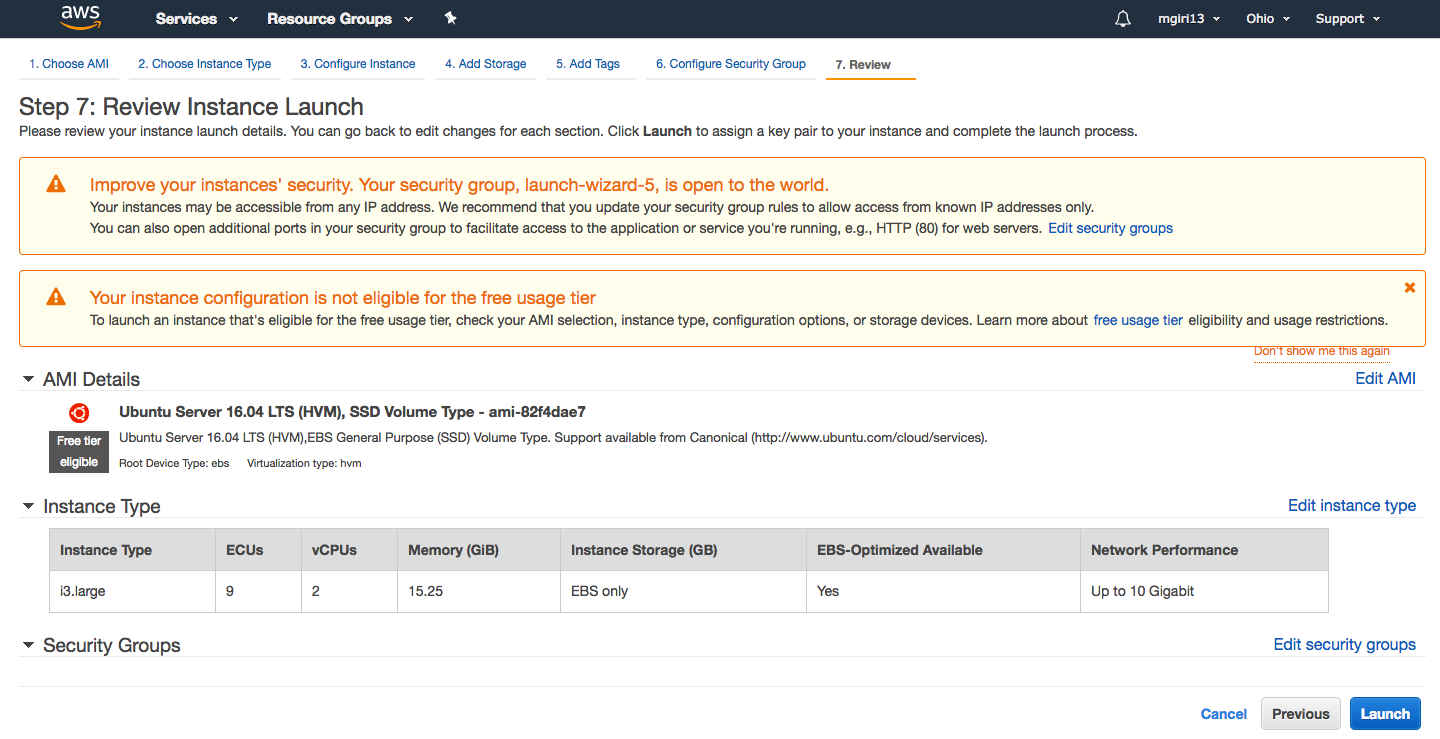
5. Add a tag for the instance and click next



6. Select the security groups to default. And click next.



7. Review all and launch



2.HADOOP

* Description Of The Problem

Implemented hadoop Tera sort to sort 128 GB and 1 TB files. 128 GB of file was sorted on the amazon instance of i3.large type and 1 TB file was sorted on the amazon instance of i3.4xlarge type and for multinode cluster we used 8 nodes of i3.large.

* Methodology

1. Launch amazon EC2 instance.
2. Install hadoop on the instance (Steps are described in the following sections)
3. Login into the hadoop cluster
4. Copy source code to master instance
5. create input file using gensort using ./gensort -a <lineNumbers> <filePath>
6. move the file to HDFS using ./hadoop dfs -copyFromLocal <input file path> /input
7. go to hadoop bin directory
8. run Hadoop jar <jar file name> class name <inputfile path> <output filepath>
9. Copy file to local from HDFS using ./hadoop dfs -copyToLocal /output <local file path>
10. Validate output using valsort ./valsort <file name>

* Runtime Environment Settings
* hadoop Runtime environment setting are as follows
* VERSION : hadoop-2.7.4
* INSTANCE TYPE : i3. Large ubuntu
* RAM : 15.25 GB
* NO. OF CORES : 2
* virtual cores STORAGE : 435 GB
* Installation Steps You Took To Setup Your Virtual Cluster And Difficulties Faced
* Generate a Key pair and save the .pem file from Amazon account.
* Launch an instance of i3.large with 500 gb general SSD volume size.
* Connect to the instance using ssh –i ec2user@your\_instance\_publicDNS Then do a scp to send mykeypair.pem and credentials to the instance location.
* Install java and Hadoop in the instance as specified in the script file . Change the config files as mentioned in the config file.
* Generate ssh keygen and paste the private key in authorized \_keys folder.
* Create an image of this instance and create other 7 nodes using the image in AWS and generate ssh keygen in each system and copy it into all other nodes in the cluster
* One node will be setup as a namenode and will have the Ip address of all datanodes in cluster under slaves folder.

3.SPARK

* Description Of The Problem

Implemented spark Tera sort to sort 128 GB and 1 TB files. 128 GB of file was sorted on the amazon instance of i3.large type and 1 TB file was sorted on the amazon instance of i3.4xlarge type. As spark in In-memory it comparatively took lot less time to sort the files.

* Methodology

1. Launch amazon EC2 instance.
2. Install spark on the instance (Steps are described in the following sections)
3. Login into the spark cluster
4. Copy source code to master instance
5. create input file using gensort using ./gensort -a <lineNumbers> <filePath>
6. move the file to HDFS using ./hadoop dfs -copyFromLocal <input file path> /input
7. go to spark bin directory
8. run command ./spark-shell -i <source code file name>
9. Copy file to local from HDFS using ./hadoop dfs -copyToLocal /output <local file path>
10. Validate output using valsort ./valsort <file name>

* Runtime Environment Settings

Spark Runtime environment setting are as follows

VERSION : Spark-2.2.0

INSTANCE TYPE : i3. Large Amazon Linux

RAM : 15.25 GB

NO. OF CORES : 2

virtual cores STORAGE : 500 GB

* Installation Steps You Took To Setup Your Virtual Cluster And Difficulties Faced

Sorting 128 GB on single i3.large node. Initially setup spark on the EC2 instance

The steps to install spark are as follows:

Pre requirement

* Generate a Key pair and save the .pem file from Amazon account.
* Launch an instance of i3.large with 500 gb general SSD volume size.
* Connect to the instance using ssh –i ec2user@your\_instance\_publicDNS Then do a scp to send mykeypair.pem and credentials to the instance location.
* Before installing spark we need to set the environment variables AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY to our Amazon EC2 access key ID and secret access key. These can be obtained from the AWS homepage by clicking Account > Security Credentials > Access Credentials.

Installation

Install the following

sudo yum install -y python python-pip git

sudo pip install apache-libcloud

sudo apt-get install default-java

git clone git://github.com/apache/spark.git -b branch-2.2

cd spark

git clone git://github.com/amplab/spark-ec2.git

cd spark-ec2

export AWS\_SECRET\_ACCESS\_KEY=JEoV3sgIGtloDamtsNrZmvjM/iEp76euWKjZUhoy export AWS\_ACCESS\_KEY\_ID=AKIAJKARGP7G22WNIJTQ

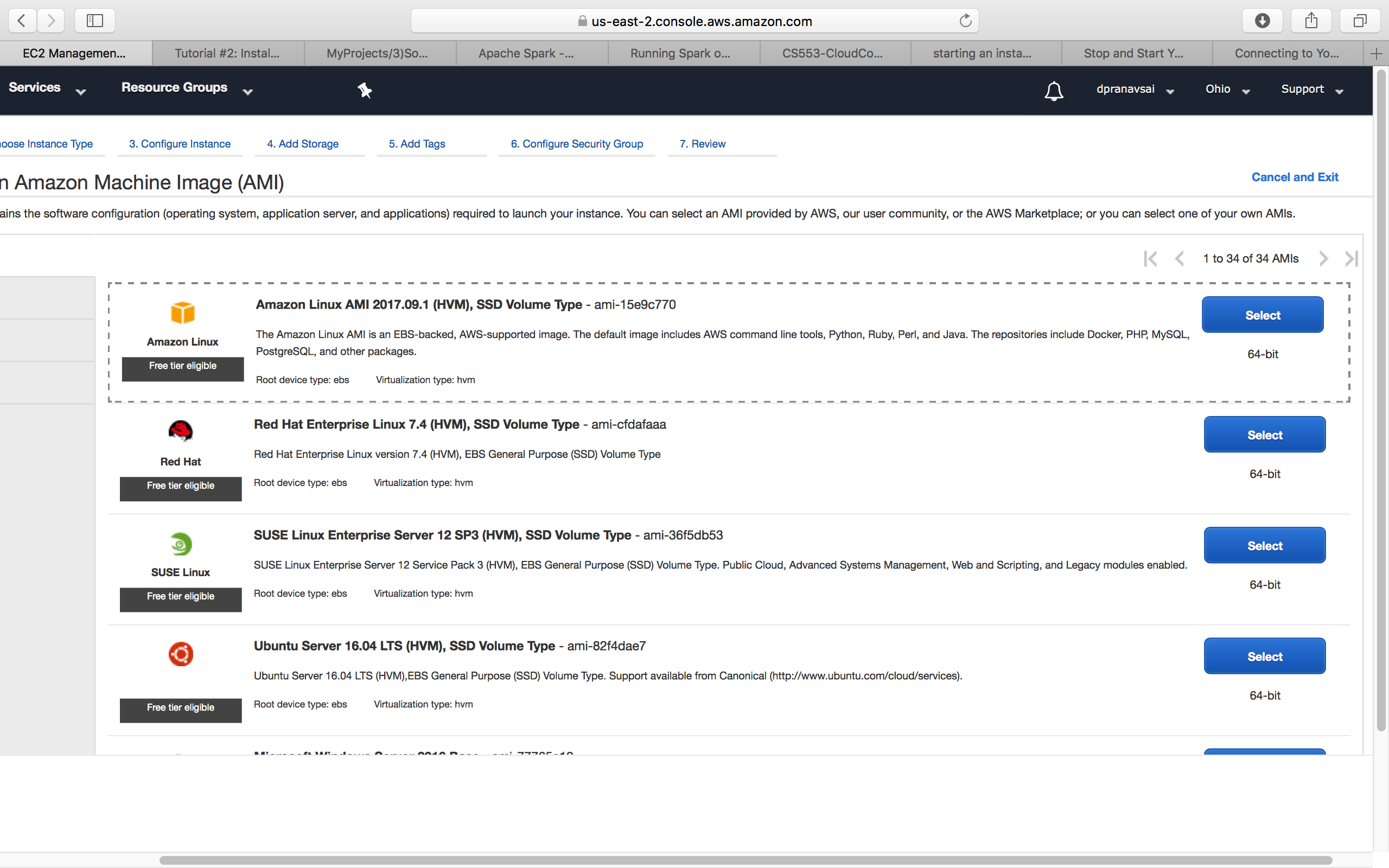
./spark-ec2 --key-pair=sparkwest2 --identity-file=sparkwest2.pem --region=us-west-2 --zone=us-west-2a --copy-aws-credentials launch myspark-cluster

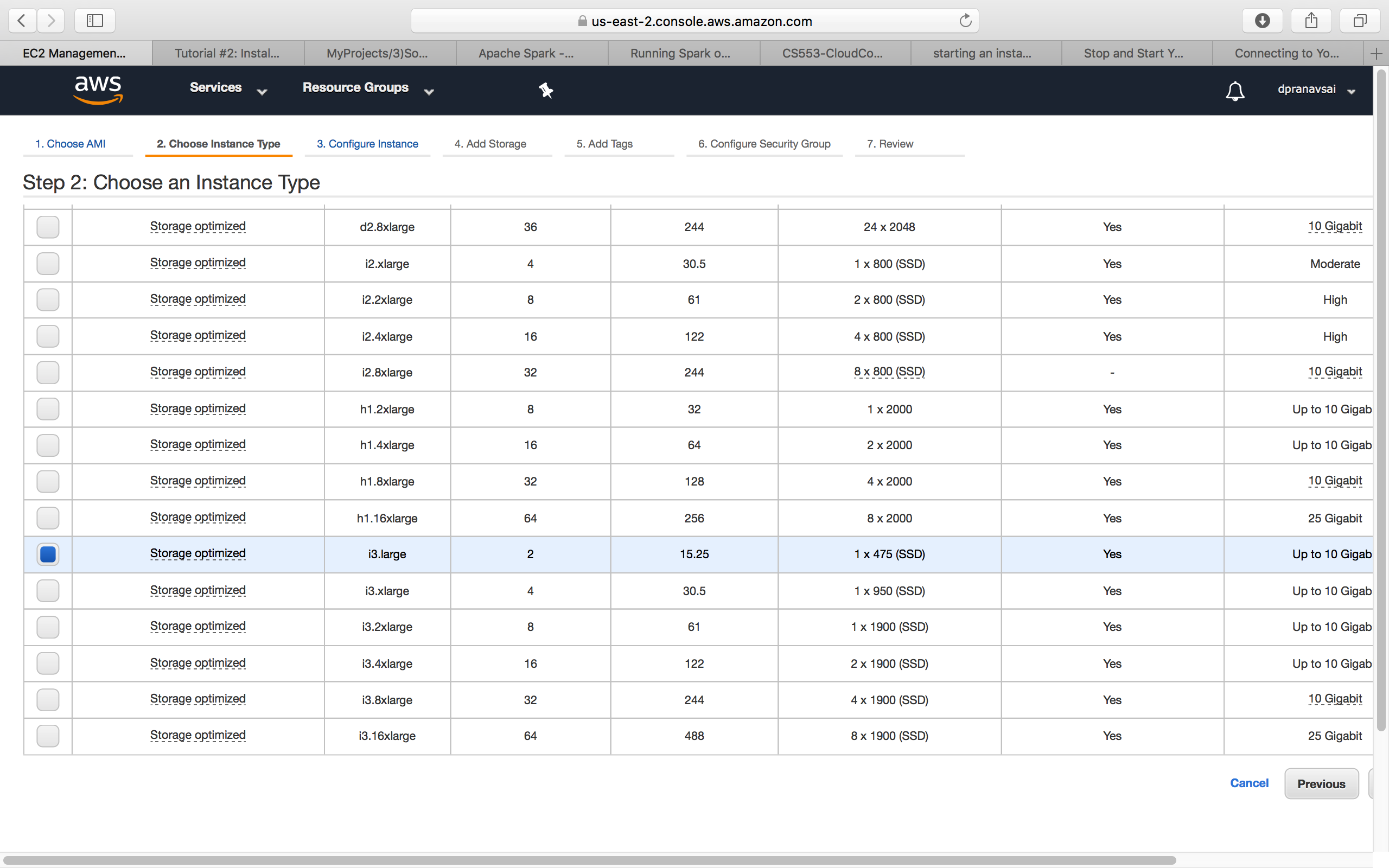
After the launch of cluster is completed login to the cluster

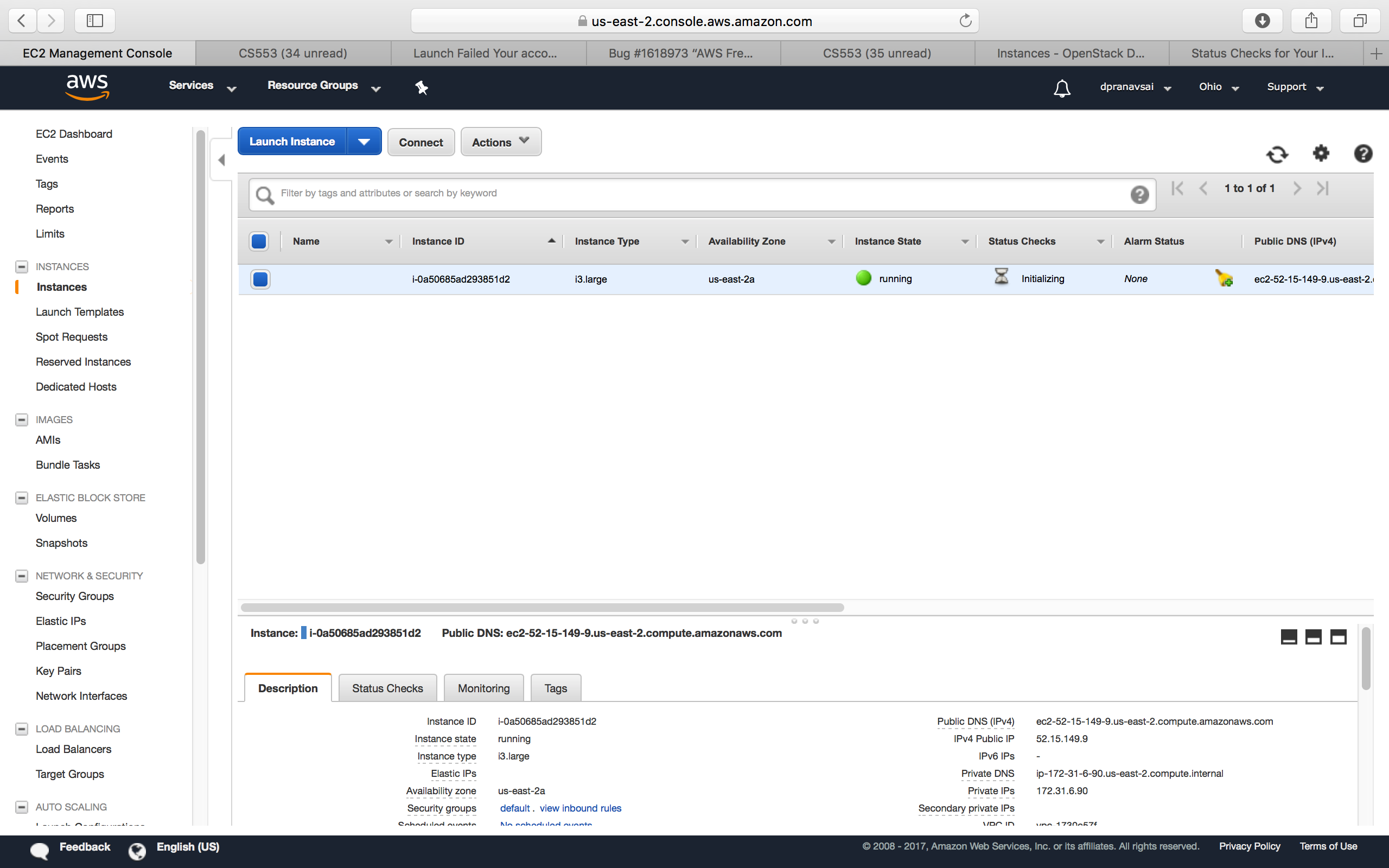
./spark-ec2 -k sparkkey -i sparkkey.pem login myspark-cluster

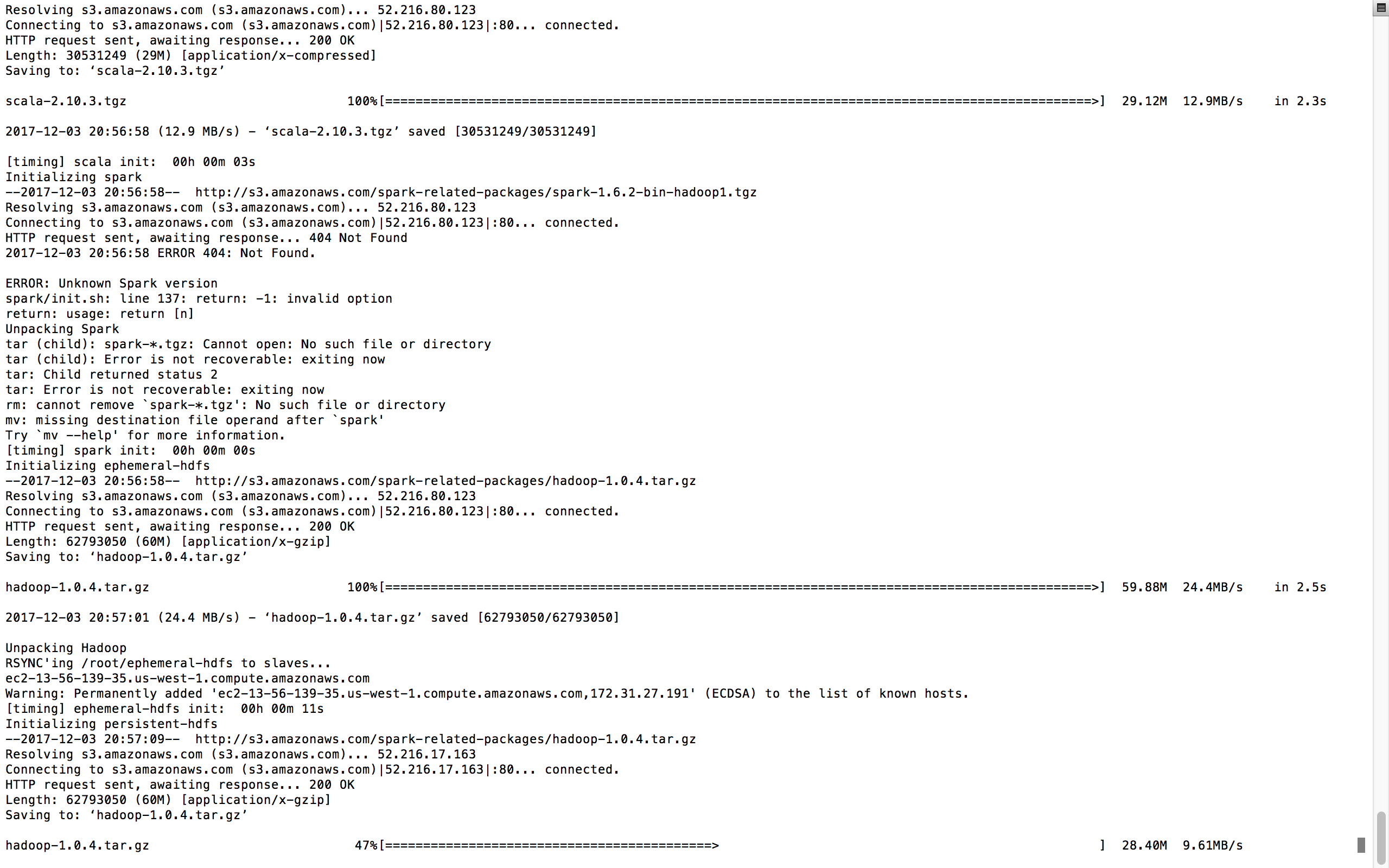
A master and slave instances with name myspark-cluster are created

Creating Instance and cluster for Spark

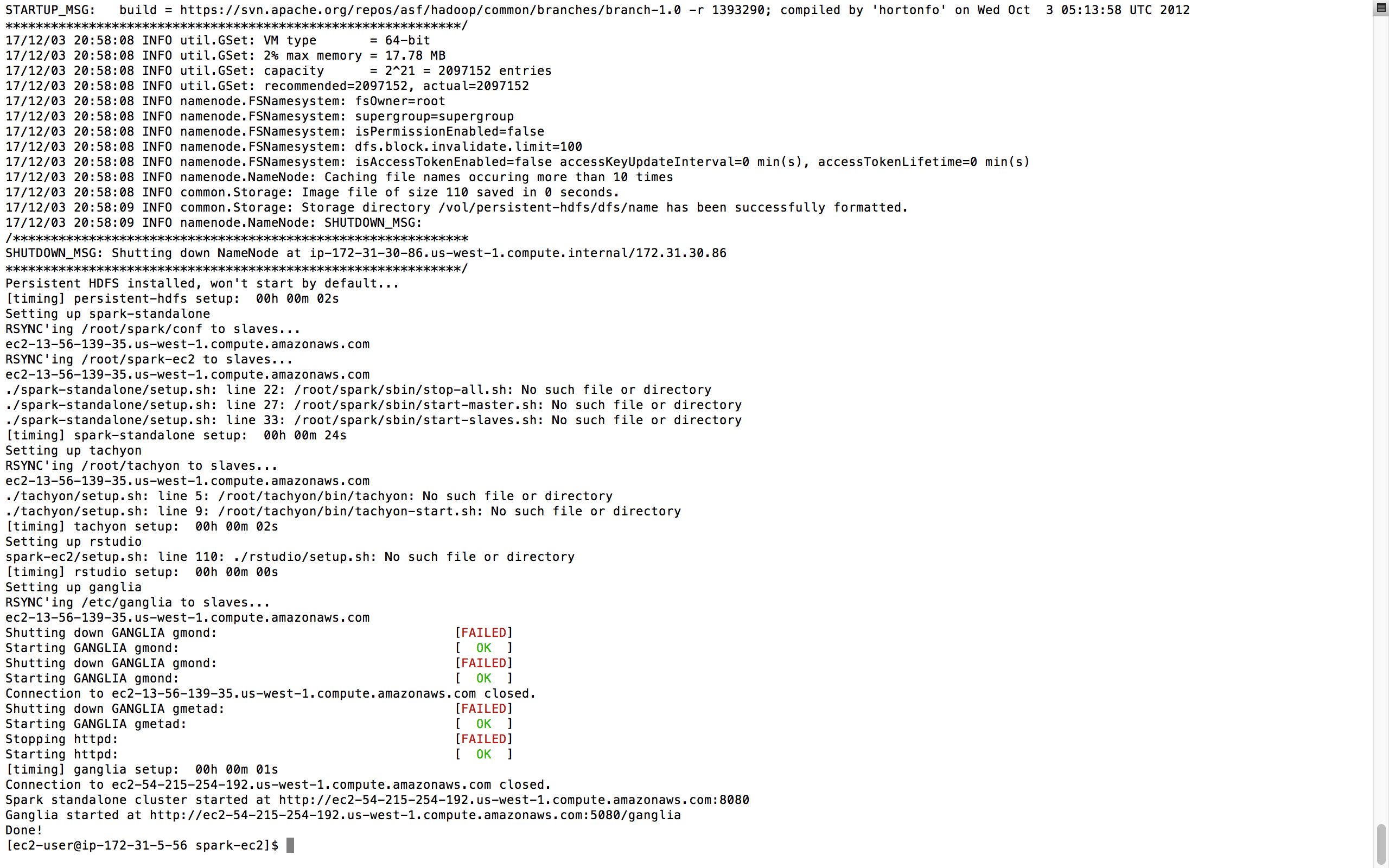


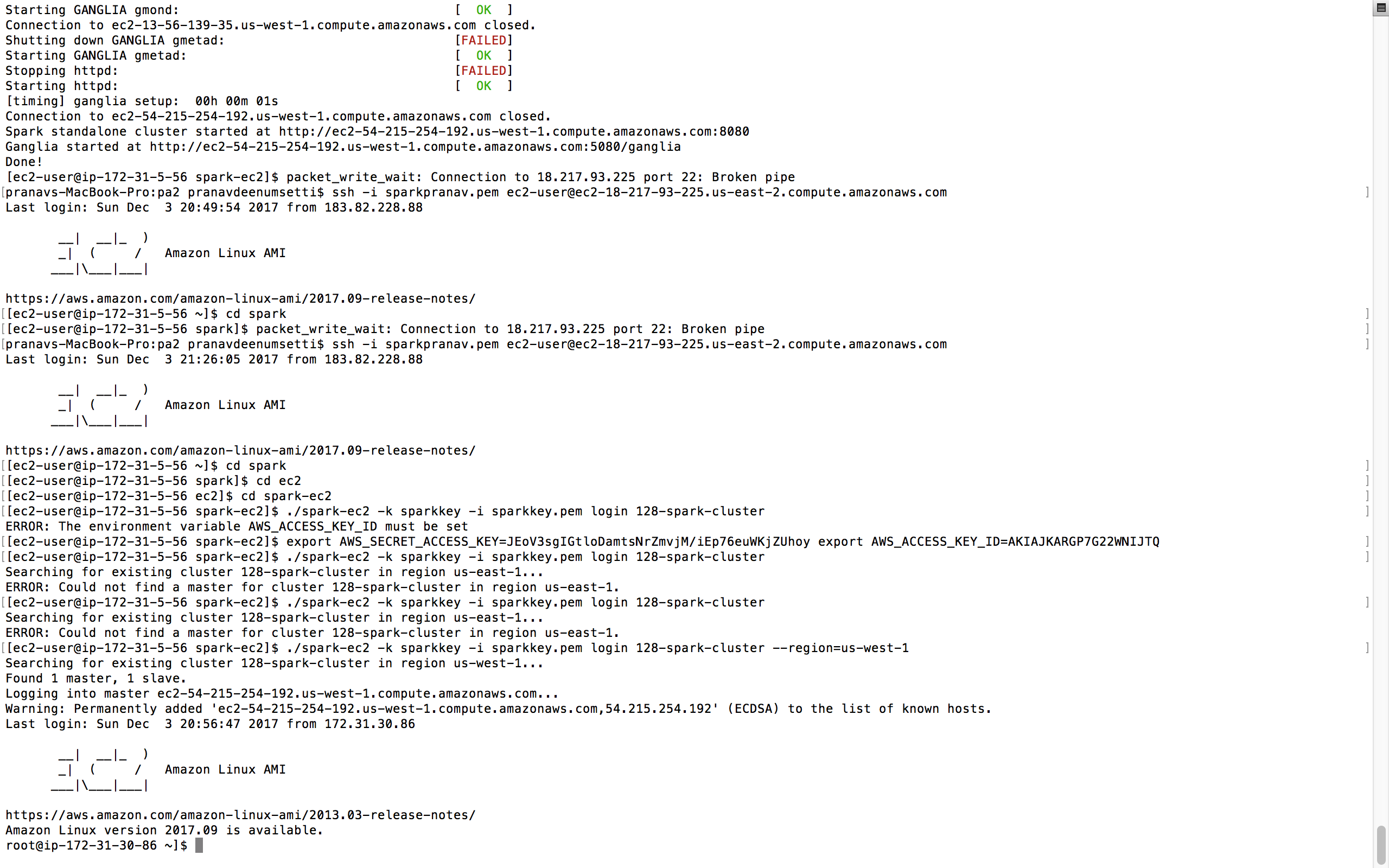






A spark cluster had been launched





Login into the spark cluster

* Detailed Description Of What Os You Used (Linux Distribution, Kernel), What Ant Version, What Java Version, What Hadoop Version, What Spark Version, And What Mpi Version You Used.

1. **Spark**

* VERSION : Spark-2.2.0
* INSTANCE TYPE : i3. Large Amazon Linux
* RAM : 15.25 GB
* NO. OF CORES : 2
* virtual cores STORAGE : 500 GB

4.TABLE

|  |  |  |  |
| --- | --- | --- | --- |
| Experiment (instance/dataset) | Shared Memory TeraSort | Hadoop TeraSort | Spark TeraSort |
| Compute Time (sec) [1xi3.large 128GB] | 24843.192 | 16283.4Sec | 12000 sec |
| Data Read (GB) [1xi3.large 128GB] | 128 | 553 GB | 298GB |
| Data Write (GB) [1xi3.large 128GB] | 128 | 681.38 GB | 336GB |
| I/O Throughput (MB/sec) [1xi3.large 128GB] | 0.00515 | 75.80 MBps | 54.10MBps |
| Compute Time (sec) [1xi3.4xlarge 1TB] | 79485.567 | 52200Sec | 45869Sec |
| Data Read (GB) [1xi3.4xlarge 1TB] | 1000 | 2895 GB | 1897GB |
| Data Write (GB) [1xi3.4xlarge 1TB] | 1000 | 3210 GB | 2148GB |
| I/O Throughput (MB/sec) [1xi3.4xlarge 1TB] | 0.012 | 119.76 MBps | 90.302 |
| Compute Time (sec) [8xi3.large 1TB] | NA | 20160sec |  |
| Data Read (GB) [8xi3.large 1TB] | NA | 2489GB |  |
| Data Write (GB) [8xi3.large 1TB] | NA | 2786GB |  |
| I/O Throughput (MB/sec) [8xi3.large 1TB] | NA | 267.93 MBps |  |
| Speedup (weak scale) |  |  |  |
| Efficiency (weak scale) |  |  |  |

5.EXPLAIN YOUR RESULTS, AND EXPLAIN THE DIFFERENCE IN PERFORMANCE?

From the table, we can infer that the time taken to sort the dataset is more in shared memory Tera sort process compared to Hadoop or Spark. In Shared memory, it involves reading the file, dividing into block size and sorting them. Later this is written to the temporary files. We should also ensure that the memory is not filled. Then we should merge all the sorted files. All this takes more time.

We can infer that for sorting 1TB Hadoop took less time in cluster when compared to single node as the compute capacity was large and number of blocks were more compared to single node system. The bottleneck faced was the storage when computing the sorting with 1TB and 128GB in single node machine. The default instance capacity was not enough. We had to mount more storage in nodes to accommodate the data. So system with more storage is required for the running the Hadoop framework approximately 4 times the storage of the data being sorted.

The same space issue was faced in spark setup as well.

REFERENCE:

<https://stackoverflow.com/questions/12807797/java-get-available-memory>

<https://stackoverflow.com/questions/17374743/how-can-i-get-the-memory-that-my-java-program-uses-via-javas-runtime-api>

<http://www.codexpedia.com/java/java-merge-sort-implementation/>

<http://www.ashishsharma.me/2011/08/external-merge-sort.html>

<http://exceptional-code.blogspot.com/2011/07/external-sorting-for-sorting-large.html>

<http://www.geeksforgeeks.org/external-sorting/>

<http://www.geeksforgeeks.org/merge-sort-for-linked-list/>

<https://spark.apache.org/docs/1.6.1/ec2-scripts.html>

<http://www.jonathanhui.com/install-hadoop-single-node-amazon-linux>

Sorted Linked List from my Data Structure course.