Running Knn Spark on EC2 Documentation

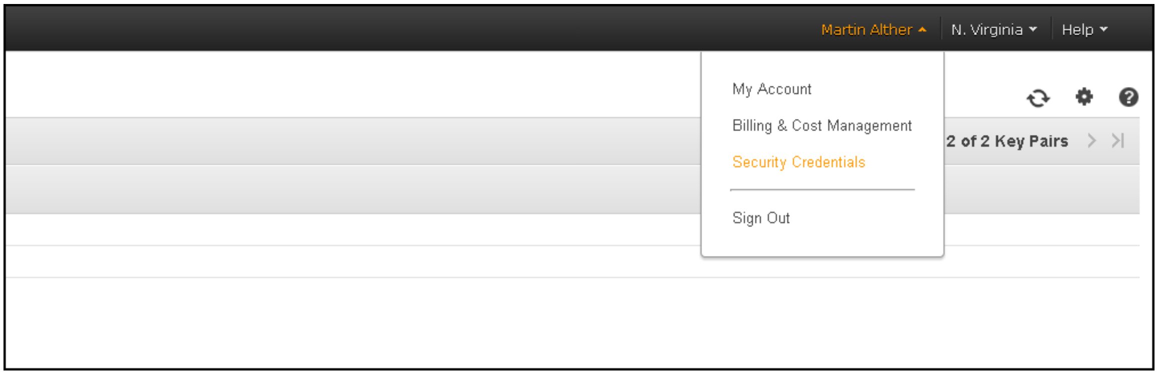
Pseudo code

```
Inputs: Train Data $D$, Test Data $X$, Number of nearest neighbours $k$
Output: Predicted class labels of $X$
Read $X$ as $RDD_X$ and $D$ from HDFS
Broadcast $D$ to all the worker nodes
Calculate the distance between each point in $RDD_X$ and $D$ as $RDD_{distance}$
Find the indices of $k$ smallest distances as nearest neighbours
Assign most frequent class label from nearest neighbours as predicted class label
Write predicted class labels to HDFS
```

Preparing to use Amazon AWS

First, open a Spark launcher instance. Open a m3.medium account with all default settings.

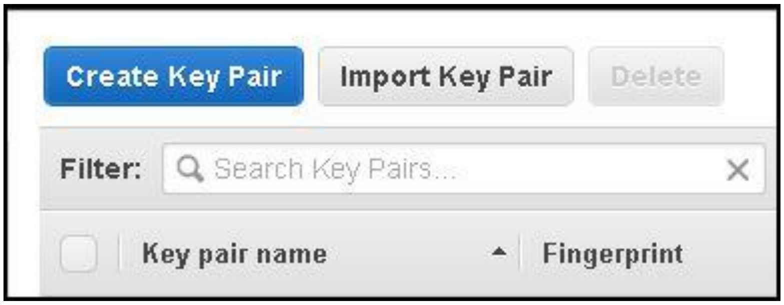
Step 1: Login to the AWS console. Under the account name at the top right, select security credentials.



Step 2: Click on the Access Keys tab and get or create the AWS access key id and AWS secret access key, then save them someplace for later.

Step 3: Under the services tab in the top left, select EC2. Once on EC2 dashboard, go to left side and click on the Network and Security tab and select Key Pairs.

Step 4: Create a new key pair, and save the key pair name as well as the .pem private key file.



Installing Spark

Step 1: Install Java

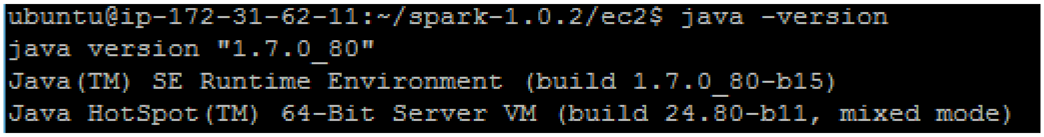
Install java with the following set of commands

- *sudo add-apt-repository ppa:webupd8team/java*
- *sudo apt-get update*
- *sudo apt-get install oracle-jdk7-installer*

You can check whether java is installed correctly by using the command

- *java –version*

If installed correctly, you should be seeing the following



```
ubuntu@ip-172-31-62-11:~/spark-1.0.2/ec2$ java -version
java version "1.7.0_80"
Java(TM) SE Runtime Environment (build 1.7.0_80-b15)
Java HotSpot(TM) 64-Bit Server VM (build 24.80-b11, mixed mode)
```

Step 2: Download and unzip Spark

Using the following commands, download a Spark version and unzip it

- *wget http://d3kbcqa49mib13.cloudfront.net/spark-1.0.2.tgz*
- *tar –xvzf spark-1.0.2.tgz*

It will create a Spark directory.

Step 3: Install Scala

Use the following commands to download Scala, install and set the path

- *wget http://www.scala-lang.org/files/archive/scala-2.10.4.tgz*
- *sudo mkdir /usr/local/src/scala*
- *sudo tar xvf scala-2.10.4.tgz -C /usr/local/src/scala/*

Open bashrc file

- *vi .bashrc*

Add the following lines at the end of the file

```
export SCALA_HOME=/usr/local/src/scala/scala-2.10.4
export PATH=$SCALA_HOME/bin:$PATH
```

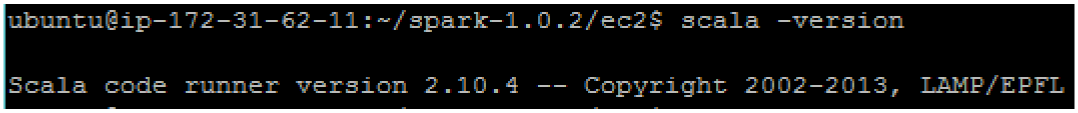
Restart bashrc by the command

- *. .bashrc*

You can check the installation by the following command

- *scala -version*

If it was installed correctly, you should be seeing the following



```
ubuntu@ip-172-31-62-11:~/spark-1.0.2/ec2$ scala -version

Scala code runner version 2.10.4 -- Copyright 2002-2013, LAMP/EPFL
```

Step 3: Install Sbt

Download and install the Simple Build Tool(sbt)

- *wget http://dl.bintray.com/sbt/debian/sbt-0.13.5.deb*
- *sudo dpkg –i sbt-0.13.5.deb*

Step 4: Build Spark

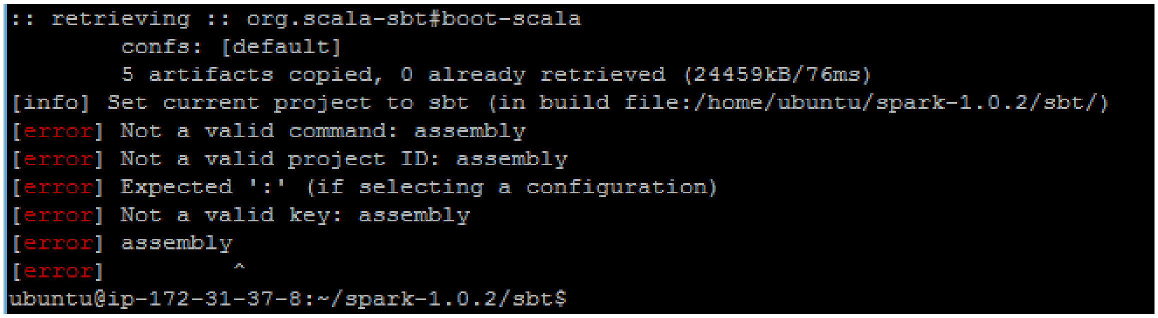
Navigate to the sbt folder inside Spark.

- *cd spark-1.0.2/sbt*

Then start building Spark by the following command

- sbt assembly

This command will install Spark. Don't worry if you see following screen with errors



```
:: retrieving :: org.scala-sbt#boot-scala
        confs: [default]
        5 artifacts copied, 0 already retrieved (24459kB/76ms)
[info] Set current project to sbt (in build file:/home/ubuntu/spark-1.0.2/sbt/)
[error] Not a valid command: assembly
[error] Not a valid project ID: assembly
[error] Expected ':' (if selecting a configuration)
[error] Not a valid key: assembly
[error] assembly
[error]         ^
ubuntu@ip-172-31-37-8:~/spark-1.0.2/sbt$
```

<u>Launching the EC2 cluster</u>

Step 1: Set environment variables for the AWS access key and secret access key that we saved in **Preparing to use AWS Step 2** with the commands:

export AWS_ACCESS_KEY_ID=<Access Key Here>
export AWS_SECRET_ACCESS_KEY=<Secret Access Key Here>

Step 2: In the Spark folder you had, navigate to the directory named "ec2".

- *cd spark-1.0.2/ec2*

Step 3:

Upload the .pem file in *spark-1.0.2/ec2* folder and change permission of the file to restricted

- *chmod 400 dmkd_spark.pem*

Step 4: Run the “spark-ec2” file with these arguments:

```
./spark-ec2 -k <keypair> -i <key-file> -s <num-slaves> --instance-type=<INSTANCE_TYPE> launch <cluster-name>
```

Where <keypair> is the name of the key pair we saved in **Preparing to use AWS Step 4**,
<key-file> is the .pem file associated with that generated key pair
<num-slaves> is the number of slave instances to launch with the master instance
<INSTANCE_TYPE> is the type of instance to be launched
and <cluster-name> is the name of the cluster we give it and will work with from now on in the EC2 scripts.

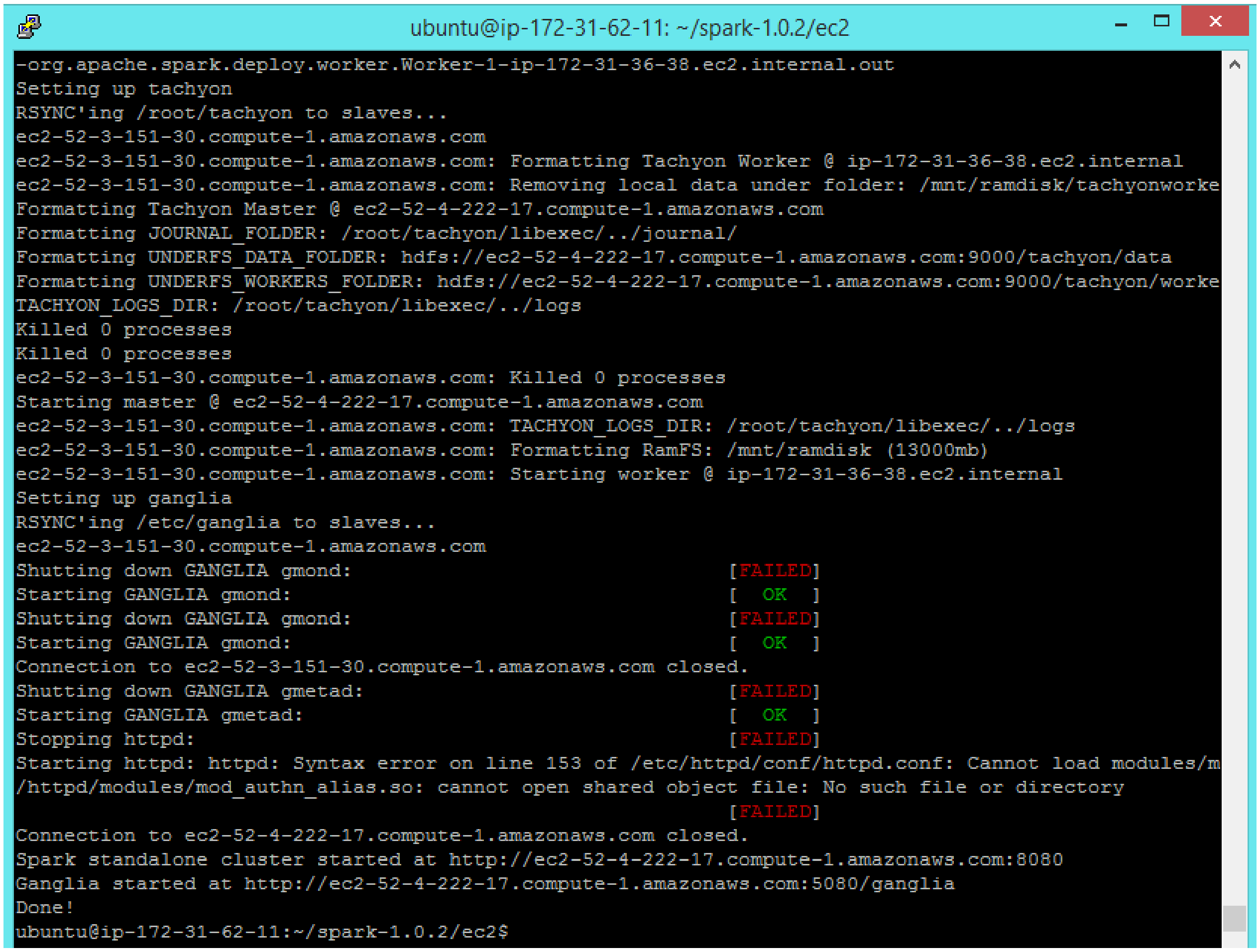
An example command is given below.

-
  ```
  ./spark-ec2 -k dmkd_spark  -i dmkd_spark.pem -s 2 --instance-type=r3.large launch spark_test
  ```

This command will create one Master and two slave instances of type r3.large. Most of the times Spark is not able to setup the cluster in first attempt due to connection refused. Try to resume with the following command

-
  ```
  ./spark-ec2 -k dmkd_spark -i dmkd_spark.pem –s 2 --instance-type=r3.large launch spark_test --resume
  ```

After Spark finish launching cluster, you should expect to see the following



```
-org.apache.spark.deploy.worker.Worker-1-ip-172-31-36-38.ec2.internal.out
Setting up tachyon
RSYNC'ing /root/tachyon to slaves...
ec2-52-3-151-30.compute-1.amazonaws.com
ec2-52-3-151-30.compute-1.amazonaws.com: Formatting Tachyon Worker @ ip-172-31-36-38.ec2.internal
ec2-52-3-151-30.compute-1.amazonaws.com: Removing local data under folder: /mnt/ramdisk/tachyonworke
Formatting Tachyon Master @ ec2-52-4-222-17.compute-1.amazonaws.com
Formatting JOURNAL_FOLDER: /root/tachyon/libexec/../journal/
Formatting UNDERFS_DATA_FOLDER: hdfs://ec2-52-4-222-17.compute-1.amazonaws.com:9000/tachyon/data
Formatting UNDERFS_WORKERS_FOLDER: hdfs://ec2-52-4-222-17.compute-1.amazonaws.com:9000/tachyon/worke
TACHYON_LOGS_DIR: /root/tachyon/libexec/../logs
Killed 0 processes
Killed 0 processes
ec2-52-3-151-30.compute-1.amazonaws.com: Killed 0 processes
Starting master @ ec2-52-4-222-17.compute-1.amazonaws.com
ec2-52-3-151-30.compute-1.amazonaws.com: TACHYON_LOGS_DIR: /root/tachyon/libexec/../logs
ec2-52-3-151-30.compute-1.amazonaws.com: Formatting RamFS: /mnt/ramdisk (13000mb)
ec2-52-3-151-30.compute-1.amazonaws.com: Starting worker @ ip-172-31-36-38.ec2.internal
Setting up ganglia
RSYNC'ing /etc/ganglia to slaves...
ec2-52-3-151-30.compute-1.amazonaws.com
Shutting down GANGLIA gmond:                               [FAILED]
Starting GANGLIA gmond:                                    [  OK  ]
Shutting down GANGLIA gmond:                               [FAILED]
Starting GANGLIA gmond:                                    [  OK  ]
Connection to ec2-52-3-151-30.compute-1.amazonaws.com closed.
Shutting down GANGLIA gmetad:                              [FAILED]
Starting GANGLIA gmetad:                                   [  OK  ]
Stopping httpd:                                            [FAILED]
Starting httpd: httpd: Syntax error on line 153 of /etc/httpd/conf/httpd.conf: Cannot load modules/m
/httpd/modules/mod_authn_alias.so: cannot open shared object file: No such file or directory
                                                           [FAILED]
Connection to ec2-52-4-222-17.compute-1.amazonaws.com closed.
Spark standalone cluster started at http://ec2-52-4-222-17.compute-1.amazonaws.com:8080
Ganglia started at http://ec2-52-4-222-17.compute-1.amazonaws.com:5080/ganglia
Done!
ubuntu@ip-172-31-62-11:~/spark-1.0.2/ec2$
```

Note down the Public DNS address of the master node (*ec2-52-4-222-17.compute-1.amazonaws.com* for the image above). We will use this to login to the cluster and run our code.

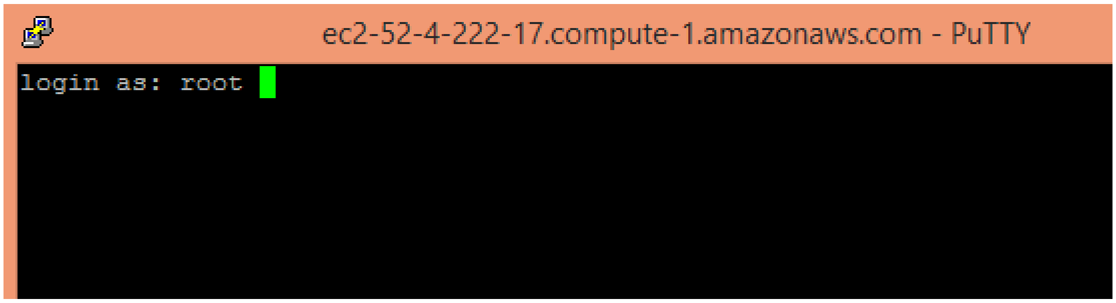
<u>Running code for Knn on the cluster</u>

Step 1: Using WinSCP login into the Spark master. Logging in is a little different from other platforms. User only the public DNS only as Host Name and put username as “root”. For example, it should be following for the Spark cluster (***spark_test***) created above

- Host name: *ec2-52-4-222-17.compute-1.amazonaws.com*
- User name: *root*

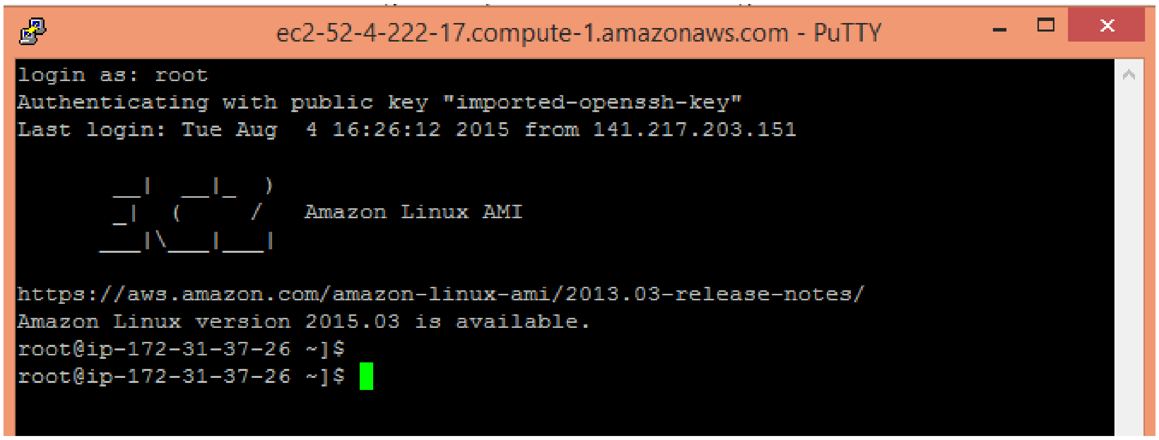
You will need to provide the KeyValue file (*dmkd_spark.ppk*) for authentication by browsing SSH->Auth.

Login to Spark Master using PuTTy with above hostname. When prompted for username, give “*root*” and press Enter



```
login as: root
```

If the login is successful, you should be seeing the following



```
login as: root
Authenticating with public key "imported-openssh-key"
Last login: Tue Aug  4 16:26:12 2015 from 141.217.203.151

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2013.03-release-notes/
Amazon Linux version 2015.03 is available.
root@ip-172-31-37-26 ~]$
root@ip-172-31-37-26 ~]$
```

Step 2: Create a folder named “*Knn*” and inside the folder upload the jar **TestKNN.jar** file and data files **train.txt** and **test.txt**

Step 3: Upload the jar file to all the other nodes in the cluster. Use the following command
./spark-ec2/copy-dir Knn

Step 4: Upload the data file in HDFS. Use the following command

- *ephemeral-hdfs/bin/hadoop fs -put Knn/test.txt /test.txt*

Step 5: Now run K-nn with the following command

- *./spark/bin/spark-submit --class org.sparkexample.KNN_Kartesian --master spark://ec2-52-4-222-17.compute-1.amazonaws.com:7077 Knn/TestKNN.jar /test.txt train.txt 3 9 5*
- Here
  ./spark/bin/spark-submit is the script to submit the jar file

--*class* is a parameter and put the class name with full package information. In our jar we have the class *org.sparkexample. KNN_Kartesian*

--*master* is a parameter and provide the public DNS of the Spark Master followed by the port number 7077. In our cluster we have this as following
spark://ec2-52-4-222-17.compute- 1.amazonaws.com:7077

Next parameter is the jar file address. *Knn/TestKNN.jar*

Next parameter is the train data file address */train.txt*

Next parameter is the test data file address */test.txt*

Next parameter is number of dimension

Next parameter is number train instances

The last parameter is number of nearest neighbor.

The command above will run K-nn algorithm on Spark cluster with number of nearest neighbours = 5 with provided train and test data.
The program will output time taken in millisecond (MS).
If you see JVM is running out of memory, you can specify driver and executing memory. Then the whole command would look like the following

```
./spark/bin/spark-submit --class org.sparkexample.Broadcast --executor-memory 10g --driver-memory 2g --master spark://ec2-54-173-178-192.compute-1.amazonaws.com:7077 Knn/TestKNN.jar /test.txt train.txt 3 9 5
```

Stopping the cluster

Step 1: Go to the Ec2 directory on your local machine from where you launched the cluster, in the terminal.

Step 2: Type the following command in the terminal

```
$ ./spark-ec2 destroy <your cluster-name>
./spark-ec2 destroy spark_test
```

Cleanup (Important)

Step 1: Logon to Amazon AWS and under Services select ‘Ec2’.

Step 2: Under the ‘Instances’ tab in the left column; click on ‘Instances’.

Step 3: Locate all your Hadoop instances and select them. On the top locate ‘Actions’ drop down button and click ‘Stop’ to stop the instances. You can start it and connect to the same settings whenever you want. If you terminate it, you have to create a new instance all together.