# Spark Tutorials

- Getting Started with Spark
- Getting Started with Spark MLlib
- Getting Started with Spark Streaming
- Getting Started with Spark SQL
- Getting Started with IntelliJ & Spark

*Dan Kikuchi*

*IBM North America Big Data Technical Team*

*April 25, 2015*

## Contents

# Lab 1 Getting Started with Spark

Apache Spark is a fast, general purpose cluster computing system for large-scale data processing and claims to process 10 to 100 times faster than MapReduce. It achieves this by leveraging aggressively-cached in-memory distributed computing. Spark is written in Scala, a functional programming language that runs in JVM threads. However, jobs can also be coded in Java or Python. It is possible to interact with Spark through interactive shells using either Scala or Python as well as through batch applications which can be written in Scala, Python or Java.

RDDs (Resilient Distributed Datasets) are a collection of elements that can be operated on in parallel and are the basic unit of data in Spark. There are two operations that can be applied to RDDs - transformations and actions. Transformations are 'lazy' and are only executed when called upon by an action. Actions trigger executions. RDDs can be also cached for faster processing.

Spark provides a stack of high-level extension tools that include Spark SQL (SQL layer on top of Spark), Spark Streaming (not true real-time streaming, but rather very small window batches), MLlib (machine learning), and GraphX (graph and graph-parallel computation).

Additional details and documentation can be found at http://spark.apache.org.

This lab uses a subset of data for taxi trips in New York City and will determine the top 10 taxi medallion numbers based on the number of trips. This will be implemented in the Spark shell as well as a Spark application in Scala. BigInsights Version 4 was used to create this tutorial and provides Spark 1.2.1 and Scala 2.10.4. The 'sbt' source build tool 0.13.7 was used to build the Spark application in this tutorial. Ant or Maven are alternative options.

There is an accompanying zip file with this tutorial - 'sparktutorials.zip'. It contains the sample data and code.

We hope that this proves to be a helpful tutorial as an introduction to Spark and Scala.

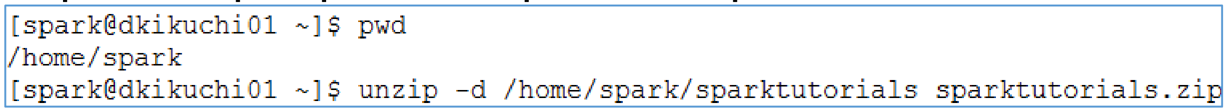
Special thanks to Mokhtar Kandil for reviewing and contributing to Lab 1.

## Part 1 Initial Set Up with Sample Data and Code

The following will setup sample data and code used in all the labs in this document.

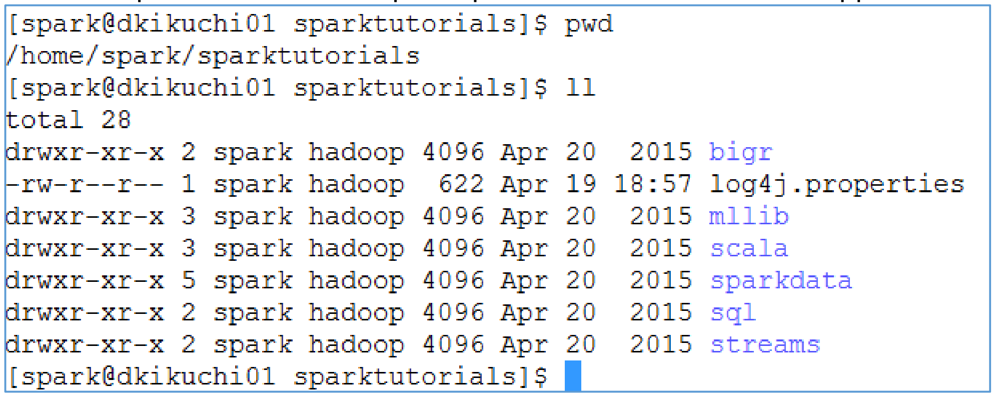
a) Please login with userid 'spark'. Alternatively, you can login as 'root' and 'su - spark'.

   Upload the file 'sparktutorials.zip' to '/home/spark' and unzip it to '/home/spark/sparktutorials'.
   **unzip -d /home/spark/sparktutorials sparktutorials.zip**



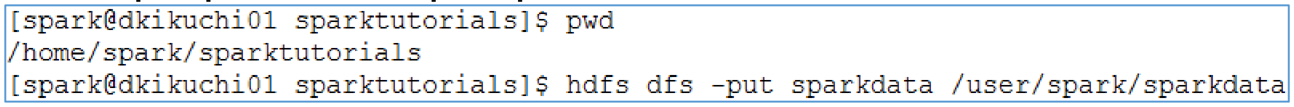
```
[spark@dkikuchi01 ~]$ pwd
/home/spark
[spark@dkikuchi01 ~]$ unzip -d /home/spark/sparktutorials sparktutorials.zip
```

Once completed, the '/home/spark/sparktutorials' folder should appear as follows.



```
[spark@dkikuchi01 sparktutorials]$ pwd
/home/spark/sparktutorials
[spark@dkikuchi01 sparktutorials]$ ll
total 28
drwxr-xr-x 2 spark hadoop 4096 Apr 20  2015 bigr
-rw-r--r-- 1 spark hadoop  622 Apr 19 18:57 log4j.properties
drwxr-xr-x 3 spark hadoop 4096 Apr 20  2015 mllib
drwxr-xr-x 3 spark hadoop 4096 Apr 20  2015 scala
drwxr-xr-x 5 spark hadoop 4096 Apr 20  2015 sparkdata
drwxr-xr-x 2 spark hadoop 4096 Apr 20  2015 sql
drwxr-xr-x 2 spark hadoop 4096 Apr 20  2015 streams
[spark@dkikuchi01 sparktutorials]$
```

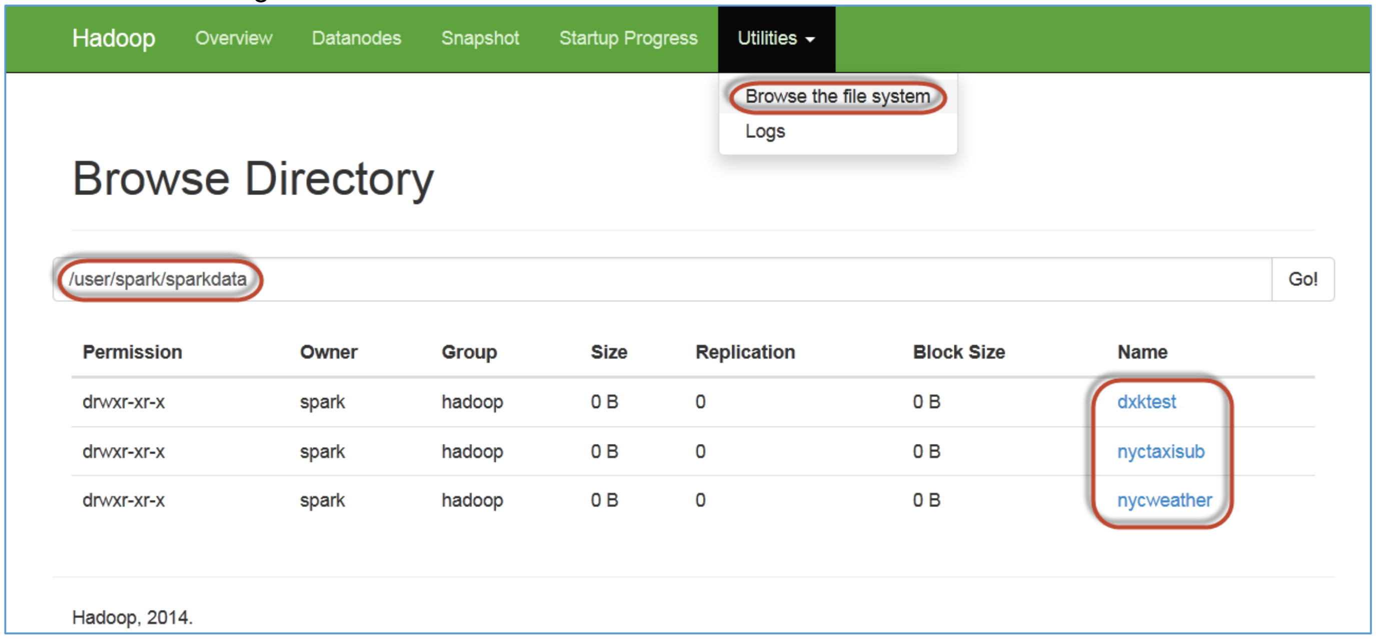
b) Move the sample data located in the 'sparkdata' folder to HDFS. From '/home/spark/sparktutorials'
**hdfs dfs -put sparkdata /user/spark/sparkdata**



```
[spark@dkikuchi01 sparktutorials]$ pwd
/home/spark/sparktutorials
[spark@dkikuchi01 sparktutorials]$ hdfs dfs -put sparkdata /user/spark/sparkdata
```

Once completed, please confirm that a '/user/spark/sparkdata/nyctaxisub/nyctaxisub.csv', '/user/spark/sparkdata/nycweather/nycweather.csv' and 'user/spark/sparkdata/dxktest/dxktest.csv' are loaded using 'hdfs dfs -ls /user/spark/sparkdata' and looking into the folders.

Alternatively, go to 'http://<yourHostName>:50070' where <yourHostName> is where the NameNode runs, click on 'Utilities' then 'Browse the file system' and navigate to '/user/spark/sparkdata' to confirm the following:



Browse Directory

/user/spark/sparkdata

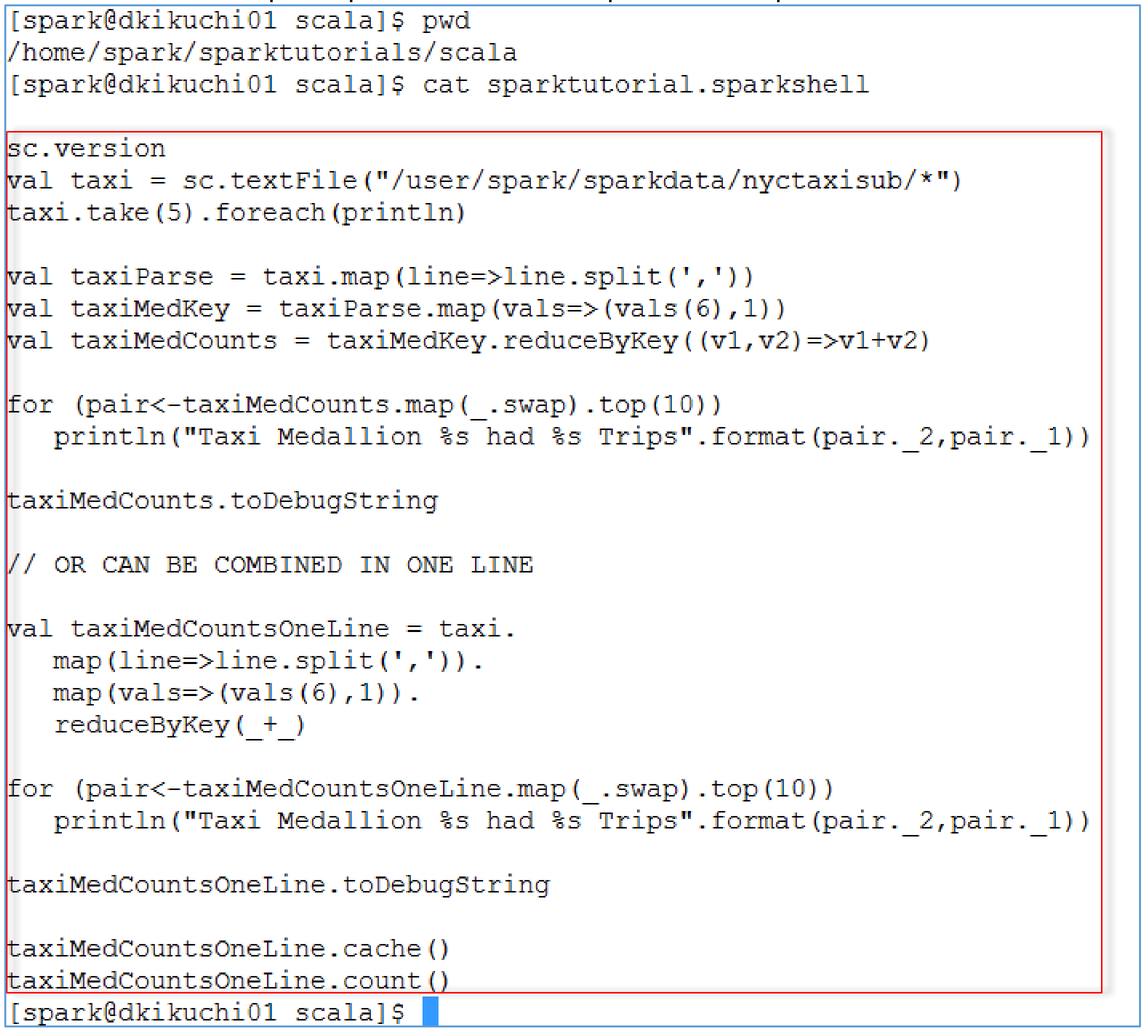
| Permission | Owner | Group | Size | Replication | Block Size | Name |
|---|---|---|---|---|---|---|
| drwxr-xr-x | spark | hadoop | 0 B | 0 | 0 B | dxktest |
| drwxr-xr-x | spark | hadoop | 0 B | 0 | 0 B | nyctaxisub |
| drwxr-xr-x | spark | hadoop | 0 B | 0 | 0 B | nycweather |

Hadoop, 2014.

## Part 2 Using the Spark Shell

In this section, the Spark shell will be used to acquire the desired results, i.e. determination of the top 10 medallion numbers based on number of trips. The sample data contains a subset of taxi trips with hack license, medallion, pickup date/time, drop off date/time, pickup latitude/longitude, drop off latitude/longitude, passenger count, trip distance, trip time and other information.

The bold commands from this document can be copied and pasted to the Spark shell, and they are also available in '/home/spark/sparktutorials/scala/sparktutorial.sparkshell'.



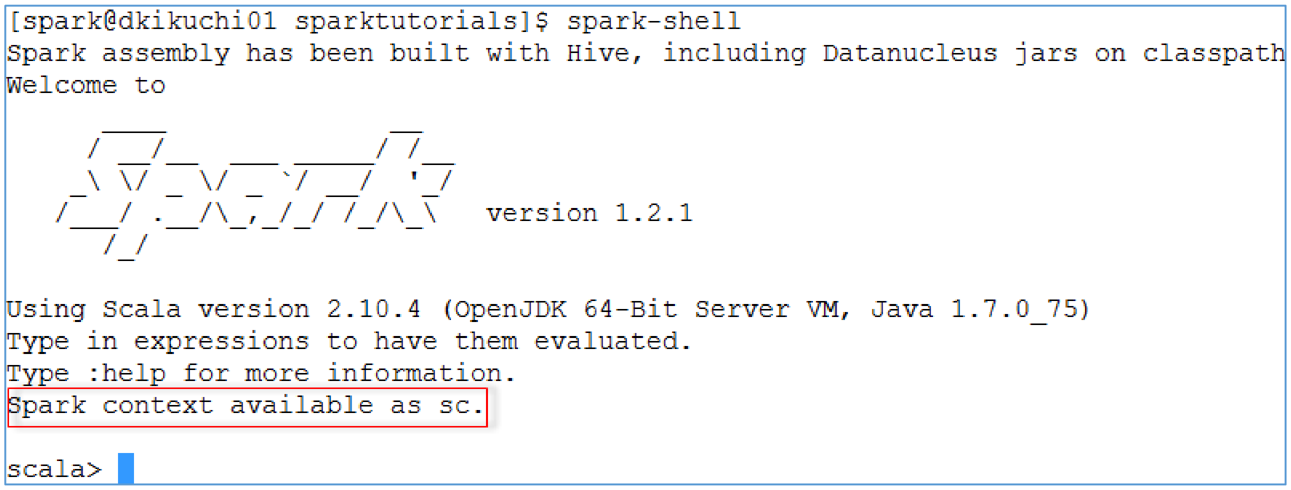
```
[spark@dkikuchi01 scala]$ pwd
/home/spark/sparktutorials/scala
[spark@dkikuchi01 scala]$ cat sparktutorial.sparkshell

sc.version
val taxi = sc.textFile("/user/spark/sparkdata/nyctaxisub/*")
taxi.take(5).foreach(println)

val taxiParse = taxi.map(line=>line.split(','))
val taxiMedKey = taxiParse.map(vals=>(vals(6),1))
val taxiMedCounts = taxiMedKey.reduceByKey((v1,v2)=>v1+v2)

for (pair<-taxiMedCounts.map(_.swap).top(10))
   println("Taxi Medallion %s had %s Trips".format(pair._2,pair._1))

taxiMedCounts.toDebugString

// OR CAN BE COMBINED IN ONE LINE

val taxiMedCountsOneLine = taxi.
   map(line=>line.split(',')).
   map(vals=>(vals(6),1)).
   reduceByKey(_+_)

for (pair<-taxiMedCountsOneLine.map(_.swap).top(10))
   println("Taxi Medallion %s had %s Trips".format(pair._2,pair._1))

taxiMedCountsOneLine.toDebugString

taxiMedCountsOneLine.cache()
taxiMedCountsOneLine.count()
[spark@dkikuchi01 scala]$
```

The inner red box shows the contents of this file.

The Spark shell can be verbose. To view ERRORs only, a 'log4j.properties' file has been provided in '/home/spark/sparktutorials'. Please launch the 'spark-shell' in this subdirectory to suppress the verbose output.

c) Please ensure that the BigInsights Version 4 environment has been started. Login in as userid 'spark' or alternatively, login in as 'root' and 'su - spark'.

d) Fire up the Spark shell from '/home/spark/sparktutorials'
**spark-shell**

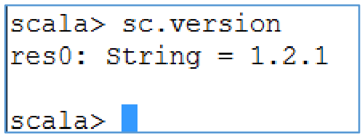


```
[spark@dkikuchi01 sparktutorials]$ spark-shell
Spark assembly has been built with Hive, including Datanucleus jars on classpath
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.2.1
      /_/

Using Scala version 2.10.4 (OpenJDK 64-Bit Server VM, Java 1.7.0_75)
Type in expressions to have them evaluated.
Type :help for more information.
Spark context available as sc.

scala>
```

Note: The version of Spark is 1.2.1 using Scala version 2.10.4 on OpenJDK 64-Bit Server VM, Java 1.7.0_75).

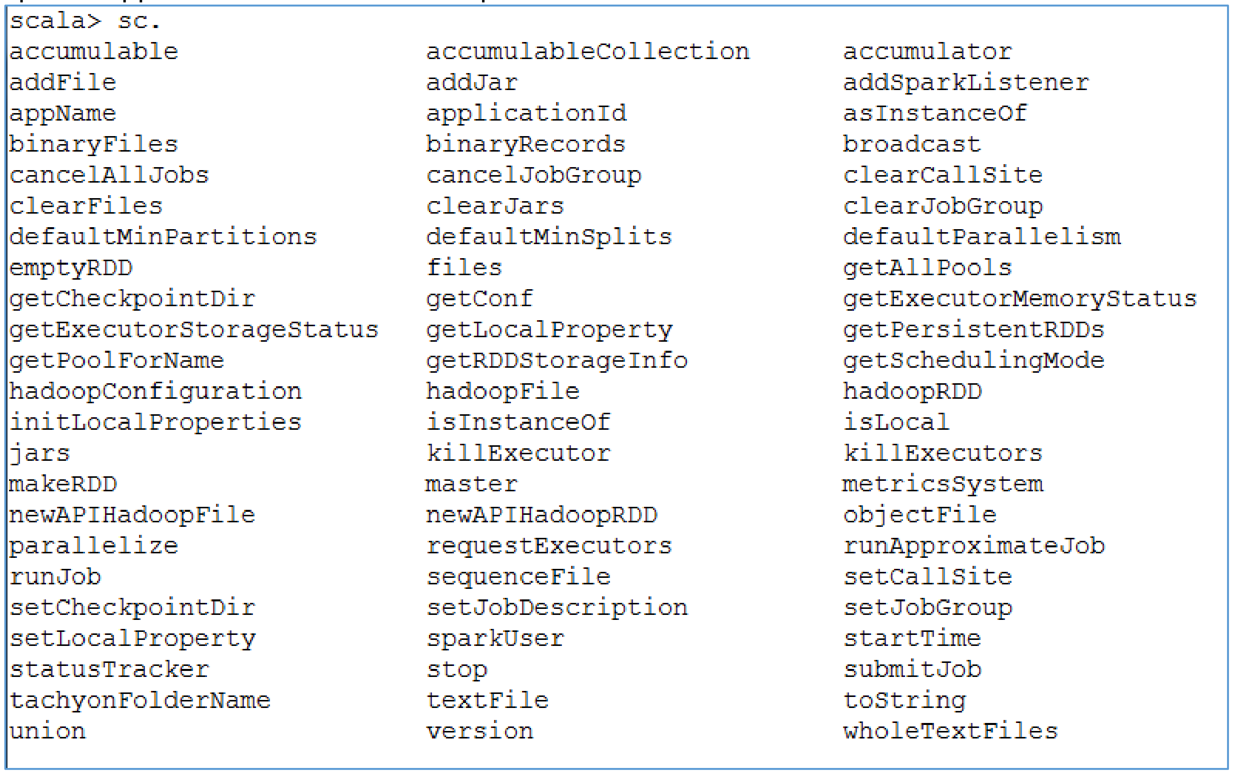
e) The shell states that the Spark context is available as ‘sc’. Using this context, the version of Spark can be confirmed.

**sc.version**



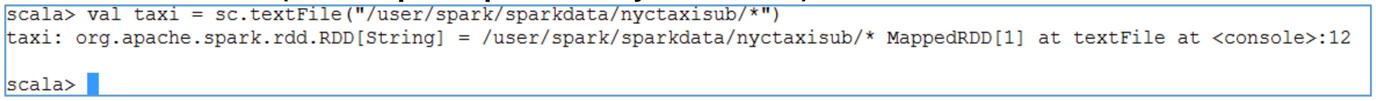
```
scala> sc.version
res0: String = 1.2.1

scala>
```

Code assist is available in the shell. For example, if ‘sc.’ is entered followed by the ‘Tab’ key, a list of options appear. A number of other options besides ‘version’ are available.



```
scala> sc.
accumulable                 accumulableCollection       accumulator
addFile                     addJar                      addSparkListener
appName                     applicationId               asInstanceOf
binaryFiles                 binaryRecords               broadcast
cancelAllJobs               cancelJobGroup              clearCallSite
clearFiles                  clearJars                   clearJobGroup
defaultMinPartitions        defaultMinSplits            defaultParallelism
emptyRDD                    files                       getAllPools
getCheckpointDir            getConf                     getExecutorMemoryStatus
getExecutorStorageStatus    getLocalProperty            getPersistentRDDs
getPoolForName              getRDDStorageInfo           getSchedulingMode
hadoopConfiguration         hadoopFile                  hadoopRDD
initLocalProperties         isInstanceOf                isLocal
jars                        killExecutor                killExecutors
makeRDD                     master                      metricsSystem
newAPIHadoopFile            newAPIHadoopRDD             objectFile
parallelize                 requestExecutors            runApproximateJob
runJob                      sequenceFile                setCallSite
setCheckpointDir            setJobDescription           setJobGroup
setLocalProperty            sparkUser                   startTime
statusTracker               stop                        submitJob
tachyonFolderName           textFile                    toString
union                       version                     wholeTextFiles
```

f) Create an RDD from the HDFS data in '/user/spark/sparkdata/nyctaxisub/nyctaxisub.csv'.
**val taxi = sc.textFile("/user/spark/sparkdata/nyctaxisub/*")**

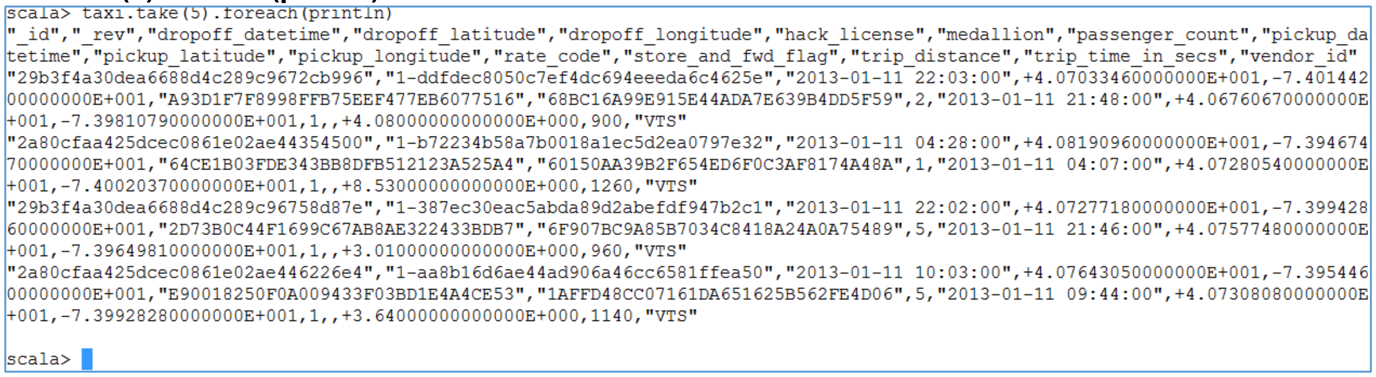


```
scala> val taxi = sc.textFile("/user/spark/sparkdata/nyctaxisub/*")
taxi: org.apache.spark.rdd.RDD[String] = /user/spark/sparkdata/nyctaxisub/* MappedRDD[1] at textFile at <console>:12

scala>
```

Keep in mind that RDDs are not processed until an action is performed.

Note: The Resilient Distributed Dataset "taxi" resulting from the command above will be composed (when an action is performed and the set is actually instantiated) of a collection of Strings corresponding to the individual lines in the file nyctaxisub.csv

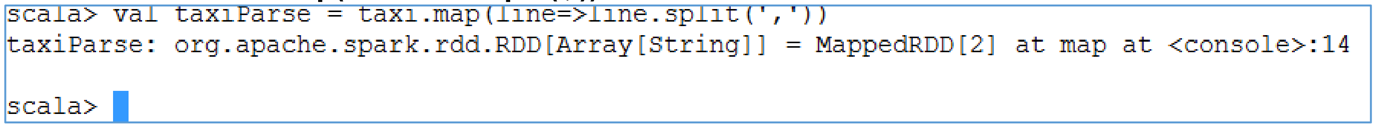
g) To view 5 rows of content, the 'take' action is invoked.
**taxi.take(5).foreach(println)**



```
scala> taxi.take(5).foreach(println)
"_id","_rev","dropoff_datetime","dropoff_latitude","dropoff_longitude","hack_license","medallion","passenger_count","pickup_da
tetime","pickup_latitude","pickup_longitude","rate_code","store_and_fwd_flag","trip_distance","trip_time_in_secs","vendor_id"
"29b3f4a30dea6688d4c289c9672cb996","1-ddfdec8050c7ef4dc694eeeda6c4625e","2013-01-11 22:03:00",+4.07033460000000E+001,-7.401442
00000000E+001,"A93D1F7F8998FFB75EEF477EB6077516","68BC16A99E915E44ADA7E639B4DD5F59",2,"2013-01-11 21:48:00",+4.06760670000000E
+001,-7.39810790000000E+001,1,,+4.08000000000000E+000,900,"VTS"
"2a80cfaa425dcec0861e02ae44354500","1-b72234b58a7b0018a1ec5d2ea0797e32","2013-01-11 04:28:00",+4.08190960000000E+001,-7.394674
70000000E+001,"64CE1B03FDE343BB8DFB512123A525A4","60150AA39B2F654ED6F0C3AF8174A48A",1,"2013-01-11 04:07:00",+4.07280540000000E
+001,-7.40020370000000E+001,1,,+8.53000000000000E+000,1260,"VTS"
"29b3f4a30dea6688d4c289c96758d87e","1-387ec30eac5abda89d2abefdf947b2c1","2013-01-11 22:02:00",+4.07277180000000E+001,-7.399428
60000000E+001,"2D73B0C44F1699C67AB8AE322433BDB7","6F907BC9A85B7034C8418A24A0A75489",5,"2013-01-11 21:46:00",+4.07577480000000E
+001,-7.39649810000000E+001,1,,+3.01000000000000E+000,960,"VTS"
"2a80cfaa425dcec0861e02ae446226e4","1-aa8b16d6ae44ad906a46cc6581ffea50","2013-01-11 10:03:00",+4.07643050000000E+001,-7.395446
00000000E+001,"E90018250F0A009433F03BD1E4A4CE53","1AFFD48CC07161DA651625B562FE4D06",5,"2013-01-11 09:44:00",+4.07308080000000E
+001,-7.39928280000000E+001,1,,+3.64000000000000E+000,1140,"VTS"

scala>
```

Note: The first row is the header. This should be filtered, but will be kept since it will not have an impact on the results.

h) To parse out the values, including the medallion number, a new RDD will be defined that will split the rows using ',' as the delimiter.
**val taxiParse = taxi.map(line=>line.split(','))**



```
scala> val taxiParse = taxi.map(line=>line.split(','))
taxiParse: org.apache.spark.rdd.RDD[Array[String]] = MappedRDD[2] at map at <console>:14

scala>
```

Scala is a functional programming language and functions are objects. Defined or anonymous functions can be passed as objects. In this case, an anonymous function is passed into 'map' which will parse comma delimited rows.

Note: As indicated in the shell upon returning from the map command, the newly defined RDD "taxiParse" is a collection of Arrays of Strings (not instantiated until the first action is taken). Each array in "taxiParse" corresponds to one String from "taxi" and the contents of each one of those arrays will be individual strings resulting from splitting the original line with the comma as the delimiter.

You can inspect a sample of the contents of "taxiParse" by using the "first()" function as follows (which would be an action causing the instantiation of this RDD and RDDs in its "lineage"):

scala> taxiParse.first()

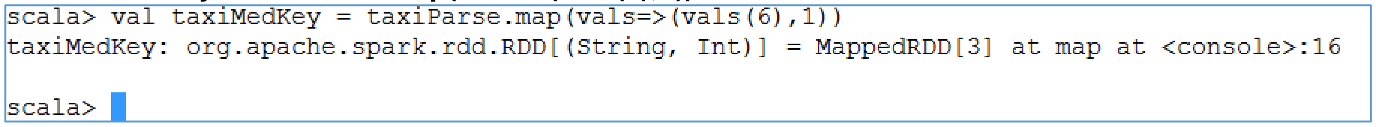
res10: Array[String] = Array("_id", "_rev", "dropoff_datetime", "dropoff_latitude", "dropoff_longitude", "hack_license", "medallion", "passenger_count", "pickup_datetime", "pickup_latitude", "pickup_longitude", "rate_code", "store_and_fwd_flag", "trip_distance", "trip_time_in_secs", "vendor_id")

The individual elements of the array can further be inspected by requesting a particular offset in it, as follows:
scala> taxiParse.first()(0)
res12: String = "_id"

or a different offset:

scala> taxiParse.first()(6)
res13: String = "medallion"

i) To create key value pairs where the key is the medallion number.
**val taxiMedKey = taxiParse.map(vals=>(vals(6),1))**



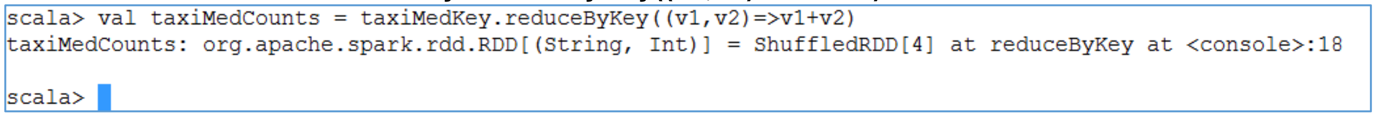
```
scala> val taxiMedKey = taxiParse.map(vals=>(vals(6),1))
taxiMedKey: org.apache.spark.rdd.RDD[(String, Int)] = MappedRDD[3] at map at <console>:16

scala> 
```

Here vals(6) is the column for the medallion number and the corresponding value is '1'. These will be counted during the 'reduceByKey'.

FYI, arrays are zero based and tuples are one based.

Note: The map operation above will define a new RDD comprised of a set of (key, value) pairs. Each key corresponds to a medallion entry (the medallion is at offset (6) in the arrays of taxiParse, as per step h) above) and the value is "1".

j) reduceByKey will process the key value pairs and count the number of occurrences for that key, which is the medallion number.
**val taxiMedCounts = taxiMedKey.reduceByKey((v1,v2)=>v1+v2)**



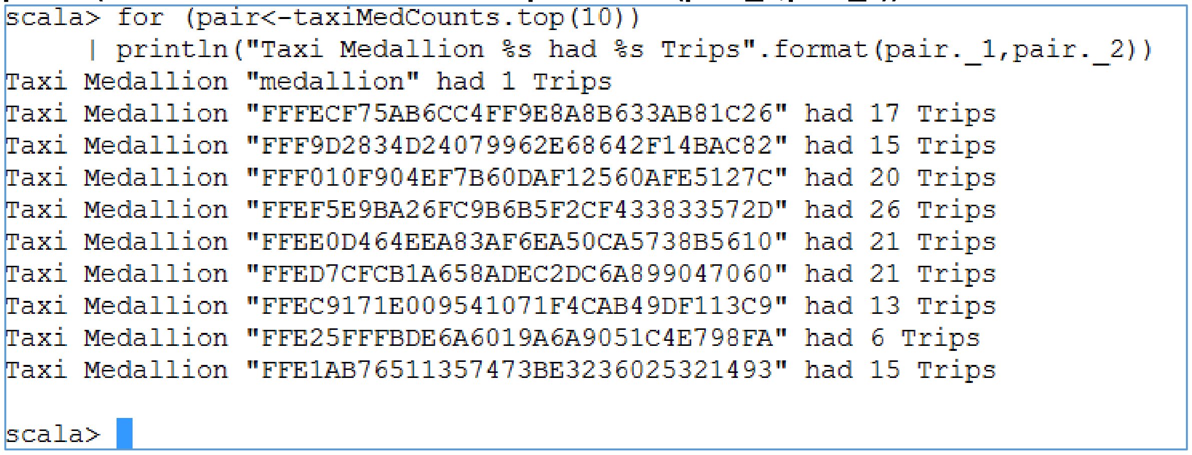
```
scala> val taxiMedCounts = taxiMedKey.reduceByKey((v1,v2)=>v1+v2)
taxiMedCounts: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[4] at reduceByKey at <console>:18

scala> 
```

Note: This transformation using "reduceByKey" will add up (when the RDD is instantiated) all the "1" values corresponding to each key (i.e to each medallion). As a result, taxiMedCounts is an RDD comprised of a set of new (key, value) pairs where the key is the "medallion" and the value is the sum of all trips for that medallion. You can inspect a sample of the contents of this new RDD by using "first()" once more:

scala> taxiMedCounts.first()
res16: (String, Int) = ("A9907052C8BBDED5079252EFE6177ECF",195)

k) Print the top 10 entries in taxiMedCounts as follows.

**for (pair<-taxiMedCounts.top(10))**
**println("Taxi Medallion %s had %s Trips".format(pair._1,pair._2))**

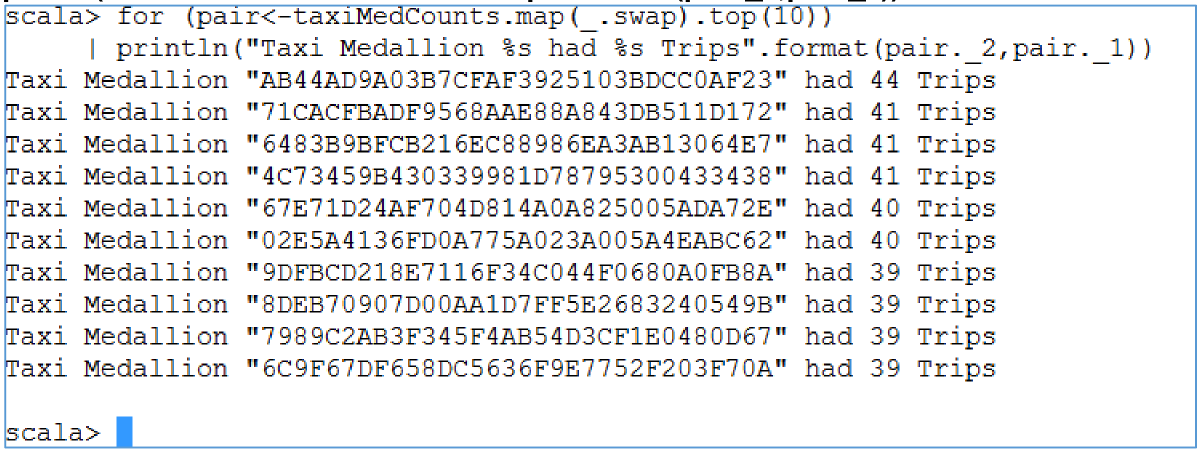


```
scala> for (pair<-taxiMedCounts.top(10))
     | println("Taxi Medallion %s had %s Trips".format(pair._1,pair._2))
Taxi Medallion "medallion" had 1 Trips
Taxi Medallion "FFFECF75AB6CC4FF9E8A8B633AB81C26" had 17 Trips
Taxi Medallion "FFF9D2834D24079962E68642F14BAC82" had 15 Trips
Taxi Medallion "FFF010F904EF7B60DAF12560AFE5127C" had 20 Trips
Taxi Medallion "FFEF5E9BA26FC9B6B5F2CF433833572D" had 26 Trips
Taxi Medallion "FFEE0D464EEA83AF6EA50CA5738B5610" had 21 Trips
Taxi Medallion "FFED7CFCB1A658ADEC2DC6A899047060" had 21 Trips
Taxi Medallion "FFEC9171E009541071F4CAB49DF113C9" had 13 Trips
Taxi Medallion "FFE25FFFBDE6A6019A6A9051C4E798FA" had 6 Trips
Taxi Medallion "FFE1AB76511357473BE3236025321493" had 15 Trips

scala>
```

However, we notice that the top 10 elements returned are organized according to the ordering of the keys in the pairs forming the RDD taxiMedCounts, (medallion, NumberOfTripsForMedallion) which are the medallion ids.

If we want to have our output sorted for the top 10 medallions **with the most trips**, it is necessary to make the number of trips the new key in the (key, value) pair and the medallion the "value". This is achieved by the function swap() which can be applied to each record in a map operation as follows:

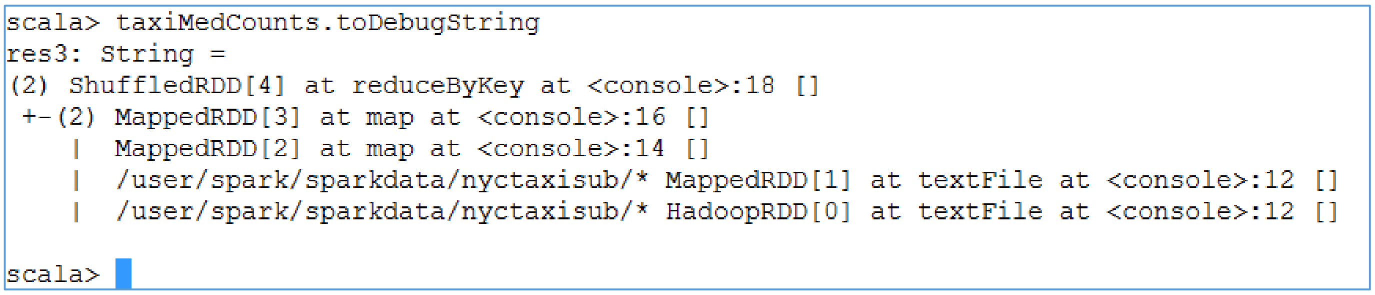
l) Now, the values are swapped and the top 10 are printed.
**for (pair<-taxiMedCounts.map(_.swap).top(10))**
**println("Taxi Medallion %s had %s Trips".format(pair._2,pair._1))**



```
scala> for (pair<-taxiMedCounts.map(_.swap).top(10))
     | println("Taxi Medallion %s had %s Trips".format(pair._2,pair._1))
Taxi Medallion "AB44AD9A03B7CFAF3925103BDCC0AF23" had 44 Trips
Taxi Medallion "71CACFBADF9568AAE88A843DB511D172" had 41 Trips
Taxi Medallion "6483B9BFCB216EC88986EA3AB13064E7" had 41 Trips
Taxi Medallion "4C73459B430339981D78795300433438" had 41 Trips
Taxi Medallion "67E71D24AF704D814A0A825005ADA72E" had 40 Trips
Taxi Medallion "02E5A4136FD0A775A023A005A4EABC62" had 40 Trips
Taxi Medallion "9DFBCD218E7116F34C044F0680A0FB8A" had 39 Trips
Taxi Medallion "8DEB70907D00AA1D7FF5E2683240549B" had 39 Trips
Taxi Medallion "7989C2AB3F345F4AB54D3CF1E0480D67" had 39 Trips
Taxi Medallion "6C9F67DF658DC5636F9E7752F203F70A" had 39 Trips

scala>
```

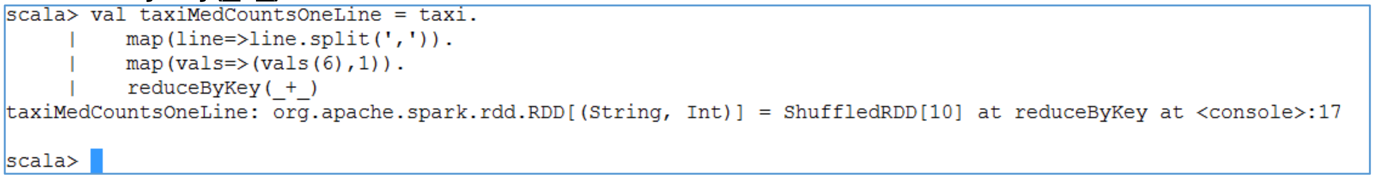
The values are swapped so that they can be ordered in descending order and the results are presented correctly according to the number of trips.

m) The RDD lineage can by viewing by looking at the RDD operator graph.
**taxiMedCounts.toDebugString**



```
scala> taxiMedCounts.toDebugString
res3: String =
(2) ShuffledRDD[4] at reduceByKey at <console>:18 []
 +-(2) MappedRDD[3] at map at <console>:16 []
    |  MappedRDD[2] at map at <console>:14 []
    |  /user/spark/sparkdata/nyctaxisub/* MappedRDD[1] at textFile at <console>:12 []
    |  /user/spark/sparkdata/nyctaxisub/* HadoopRDD[0] at textFile at <console>:12 []

scala>
```

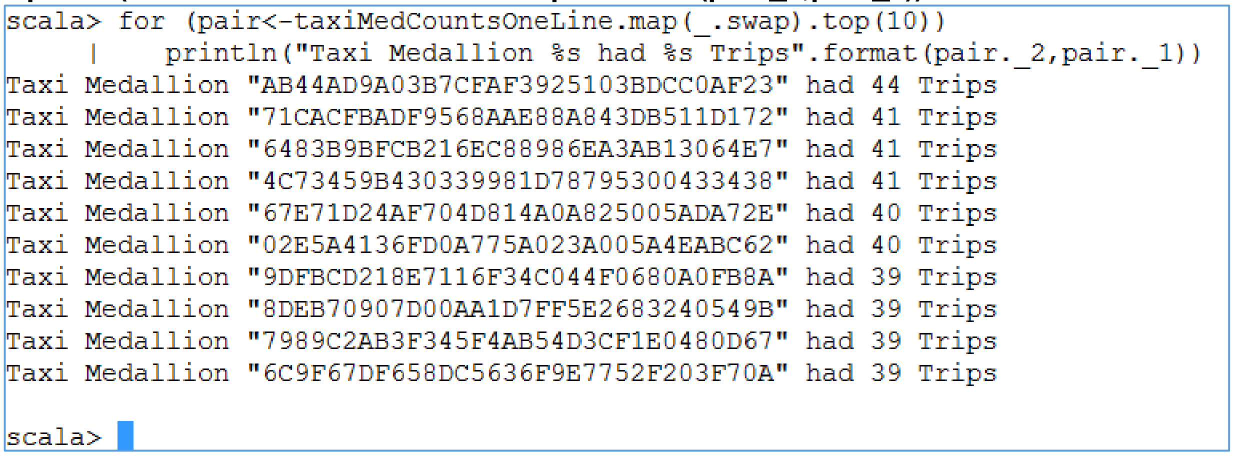
n) While each step above was done one line at a time, they could be merged into one equivalent line.
**val taxiMedCountsOneLine = taxi.**
  **map(line=>line.split(',')).**
  **map(vals=>(vals(6),1)).**
  **reduceByKey(_+_)**



```
scala> val taxiMedCountsOneLine = taxi.
     |    map(line=>line.split(',')).
     |    map(vals=>(vals(6),1)).
     |    reduceByKey(_+_)
taxiMedCountsOneLine: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[10] at reduceByKey at <console>:17

scala>
```

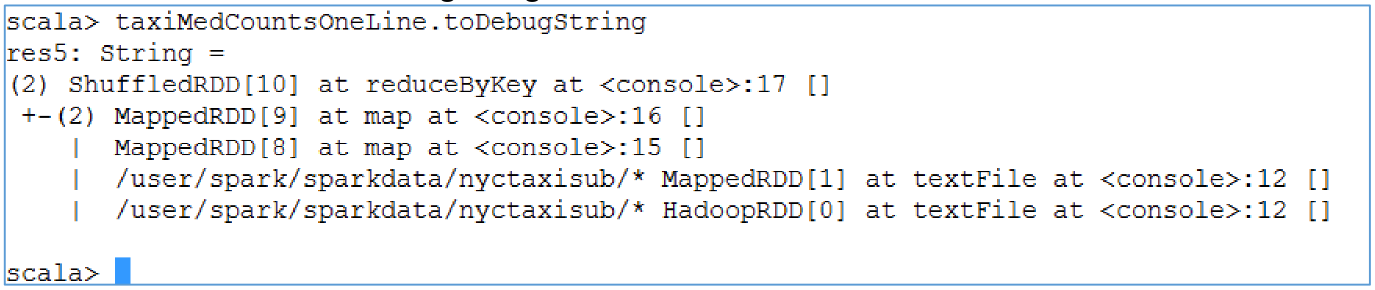
Note: reduceByKey(_+_) is a short hand equivalent to reduceByKey((v1,v2)=>v1+v2)

o) Rerun the previous println code to obtain the results as before.
**for (pair<-taxiMedCountsOneLine.map(_.swap).top(10))**
  **println("Taxi Medallion %s had %s Trips".format(pair._2,pair._1))**



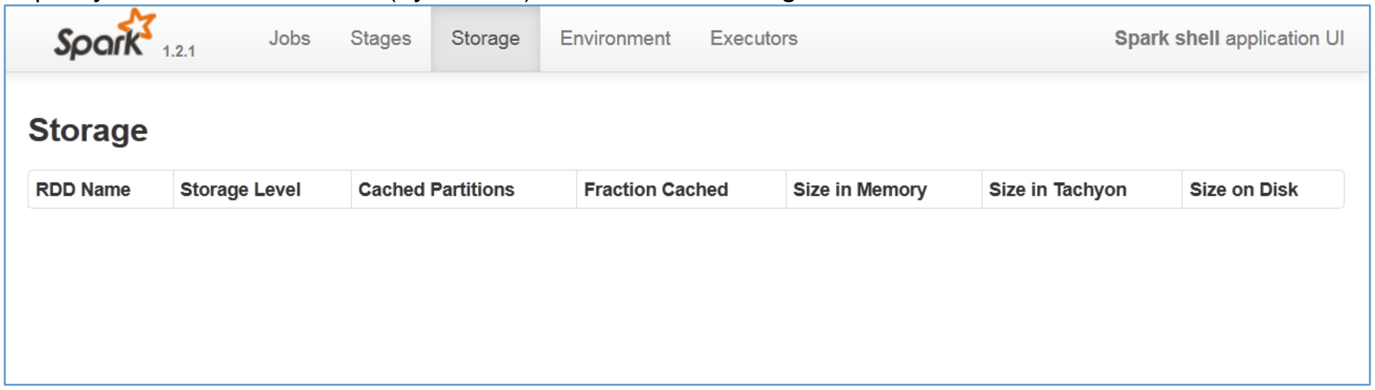
```
scala> for (pair<-taxiMedCountsOneLine.map(_.swap).top(10))
     |    println("Taxi Medallion %s had %s Trips".format(pair._2,pair._1))
Taxi Medallion "AB44AD9A03B7CFAF3925103BDCC0AF23" had 44 Trips
Taxi Medallion "71CACFBADF9568AAE88A843DB511D172" had 41 Trips
Taxi Medallion "6483B9BFCB216EC88986EA3AB13064E7" had 41 Trips
Taxi Medallion "4C73459B430339981D78795300433438" had 41 Trips
Taxi Medallion "67E71D24AF704D814A0A825005ADA72E" had 40 Trips
Taxi Medallion "02E5A4136FD0A775A023A005A4EABC62" had 40 Trips
Taxi Medallion "9DFBCD218E7116F34C044F0680A0FB8A" had 39 Trips
Taxi Medallion "8DEB70907D00AA1D7FF5E2683240549B" had 39 Trips
Taxi Medallion "7989C2AB3F345F4AB54D3CF1E0480D67" had 39 Trips
Taxi Medallion "6C9F67DF658DC5636F9E7752F203F70A" had 39 Trips

scala>
```

p) Again, the RDD lineage can be viewed by looking at the RDD operator graph.
**taxiMedCountsOneLine.toDebugString**



```
scala> taxiMedCountsOneLine.toDebugString
res5: String =
(2) ShuffledRDD[10] at reduceByKey at <console>:17 []
 +-(2) MappedRDD[9] at map at <console>:16 []
    |  MappedRDD[8] at map at <console>:15 []
    |  /user/spark/sparkdata/nyctaxisub/* MappedRDD[1] at textFile at <console>:12 []
    |  /user/spark/sparkdata/nyctaxisub/* HadoopRDD[0] at textFile at <console>:12 []

scala>
```

q) Now let's look at caching of RDDs. First, take a look at the Spark 'Storage' utilization by going to http://<yourHostName>:4040 (by default) and click on 'Storage'.

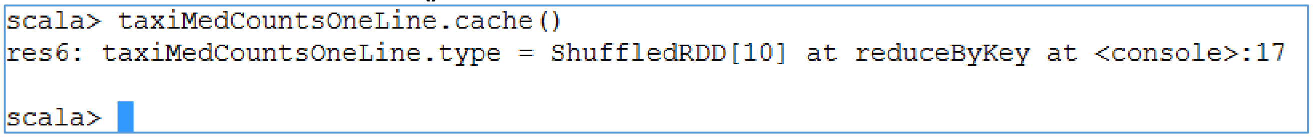


Storage

| RDD Name | Storage Level | Cached Partitions | Fraction Cached | Size in Memory | Size in Tachyon | Size on Disk |
|---|---|---|---|---|---|---|

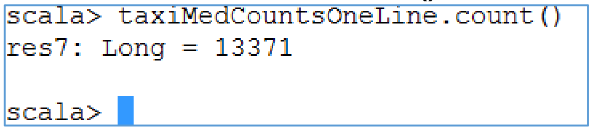
Note: Storage is not currently used.

r) Issue the following command to cache the 'taxiMedCountsOneLine' RDD.
**taxiMedCountsOneLine.cache()**



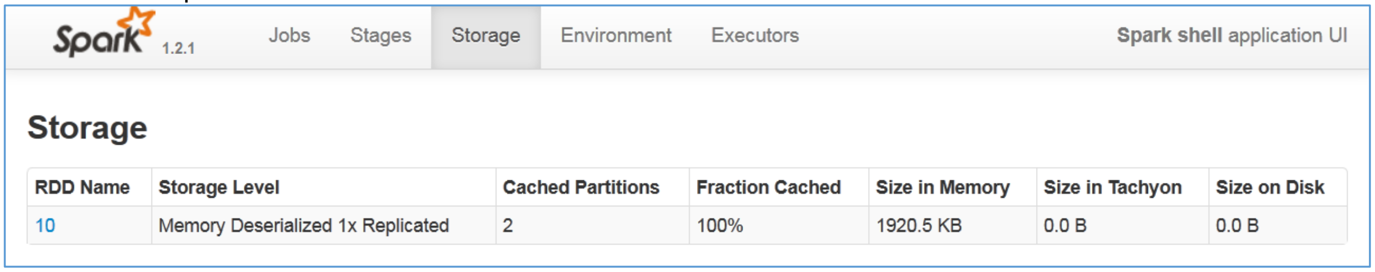
```
scala> taxiMedCountsOneLine.cache()
res6: taxiMedCountsOneLine.type = ShuffledRDD[10] at reduceByKey at <console>:17

scala>
```

s) The RDD won't be cached until an action is executed on it. 'count()' is an action that will invoke an action and caching for this RDD.
**taxiMedCountsOneLine.count()**



```
scala> taxiMedCountsOneLine.count()
res7: Long = 13371

scala>
```

t) Refresh the Spark 'Storage' browser screen and the storage information is displayed for this cached RDD. Subsequent calls on this RDD will be faster.

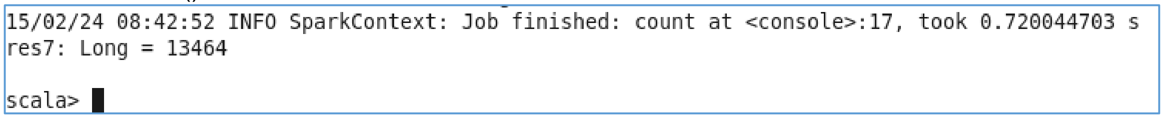


Storage

| RDD Name | Storage Level | Cached Partitions | Fraction Cached | Size in Memory | Size in Tachyon | Size on Disk |
|---|---|---|---|---|---|---|
| 10 | Memory Deserialized 1x Replicated | 2 | 100% | 1920.5 KB | 0.0 B | 0.0 B |

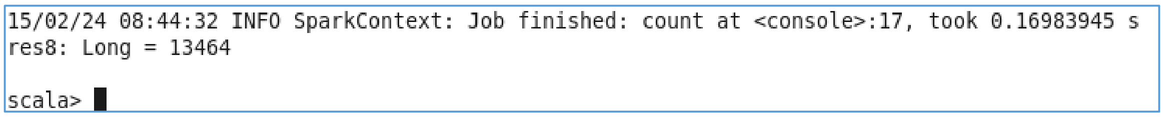
To empirically prove this point, the 'log4j.rootCategory' in 'log4j.properties' can be changed to 'INFO'. Execution times will be included in the verbose messages.

For the 'count()' after the RDD cache was set, it took 0.720044703 seconds to execute.



```
15/02/24 08:42:52 INFO SparkContext: Job finished: count at <console>:17, took 0.720044703 s
res7: Long = 13464

scala>
```

After caching, the 'count()' took 0.16983945 seconds to execute.



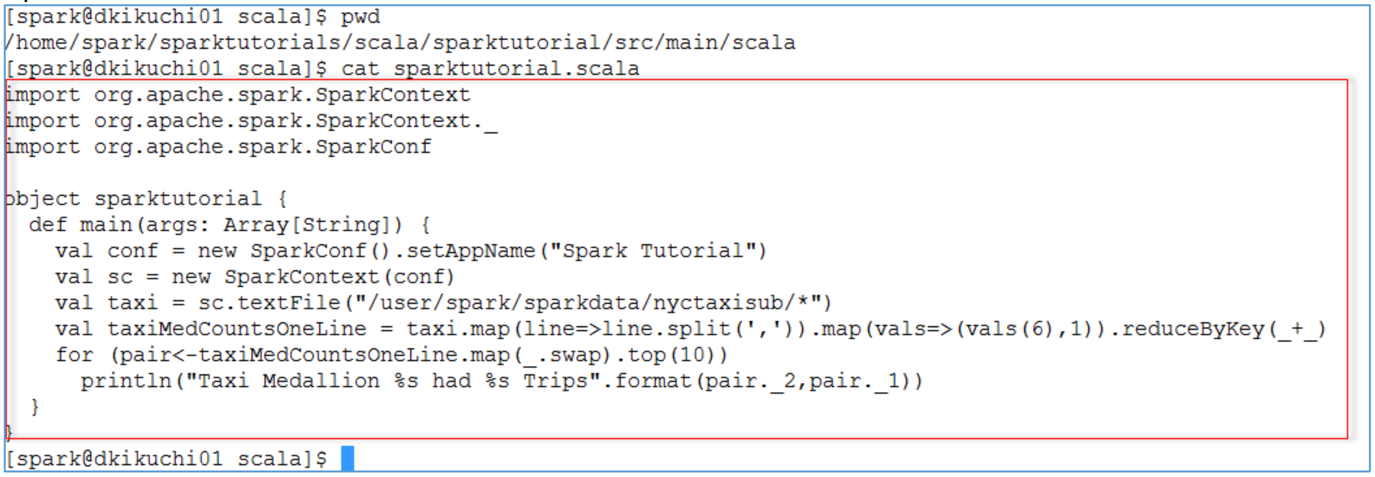
```
15/02/24 08:44:32 INFO SparkContext: Job finished: count at <console>:17, took 0.16983945 s
res8: Long = 13464

scala>
```

u) Exit the Spark shell by typing **exit** and enter.

## Part 3 Creating a Spark Application

In this section, a Spark application will be created to acquire the desired results.

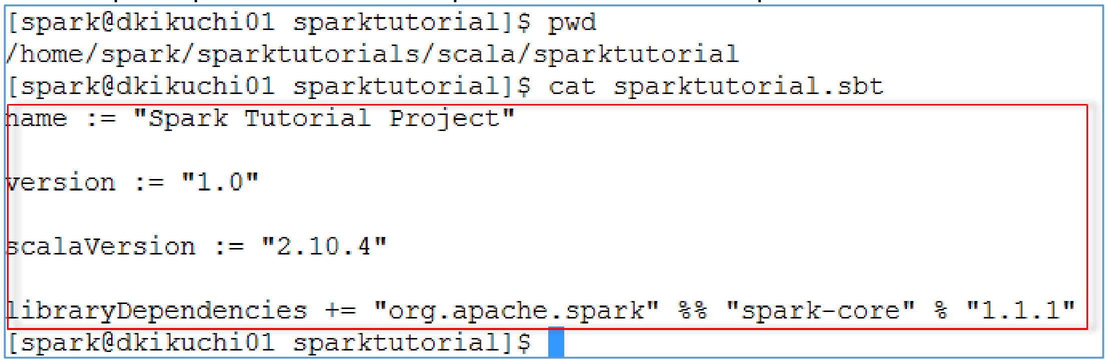
v) Please go to '/home/spark/sparktutorials/scala/sparktutorial/src/main/scala' and edit the 'sparktutorial.scala' file.



```
[spark@dkikuchi01 scala]$ pwd
/home/spark/sparktutorials/scala/sparktutorial/src/main/scala
[spark@dkikuchi01 scala]$ cat sparktutorial.scala
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object sparktutorial {
  def main(args: Array[String]) {
    val conf = new SparkConf().setAppName("Spark Tutorial")
    val sc = new SparkContext(conf)
    val taxi = sc.textFile("/user/spark/sparkdata/nyctaxisub/*")
    val taxiMedCountsOneLine = taxi.map(line=>line.split(',')).map(vals=>(vals(6),1)).reduceByKey(_+_)
    for (pair<-taxiMedCountsOneLine.map(_.swap).top(10))
      println("Taxi Medallion %s had %s Trips".format(pair._2,pair._1))
  }
}
[spark@dkikuchi01 scala]$
```

The actual code is in the inner red box. Three package imports are performed. In the 'main', the SparkContext is created and assigned to 'sc'. The code following this should look familiar as these are the commands issued in the previous steps for the Spark shell implementation. The results will be written to the log in this case, but it could also be written to HDFS.

w) 'sbt' 0.13.7 was installed and used for the build, however Maven or Ant could be used. Go to '/home/spark/sparktutorials/scala/sparktutorial' and view 'sparktutorial.sbt'.

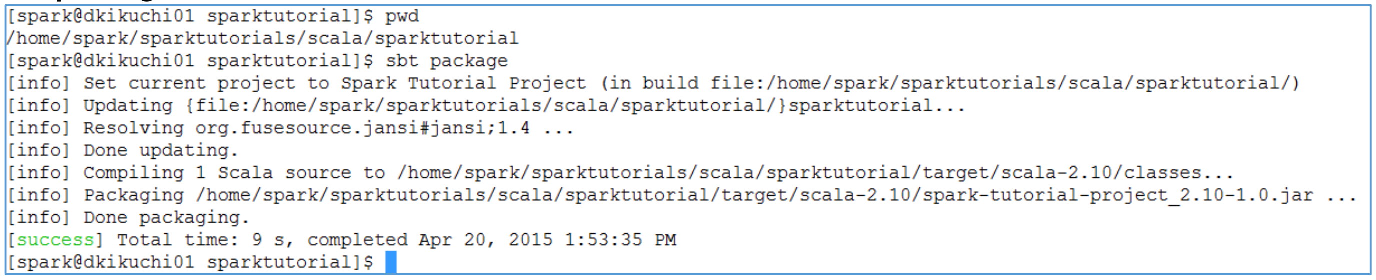


```
[spark@dkikuchi01 sparktutorial]$ pwd
/home/spark/sparktutorials/scala/sparktutorial
[spark@dkikuchi01 sparktutorial]$ cat sparktutorial.sbt
name := "Spark Tutorial Project"

version := "1.0"

scalaVersion := "2.10.4"

libraryDependencies += "org.apache.spark" %% "spark-core" % "1.1.1"
[spark@dkikuchi01 sparktutorial]$
```

This provides the name, versions and library dependencies for the build tool.

x) (OPTIONAL) In order to compile this source, 'sbt' will need to be downloaded and installed. The build tool is licensed and located at 'http://www.scala-sbt.org'. However, the source file has been pre-compiled and available. If you do not wish to install 'sbt', please go on to the next step.

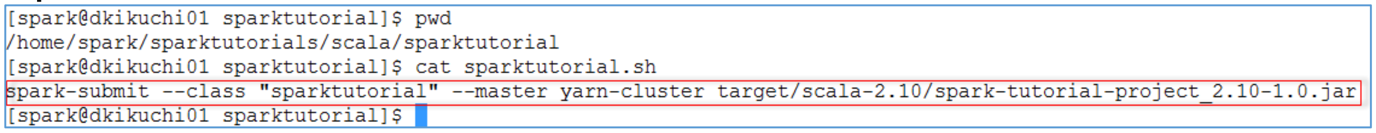
Otherwise, to re-compile the code, download and install ‘sbt’, go to ‘/home/spark/sparktutorials/scala/sparktutorial’ and issue.
**sbt package**



```
[spark@dkikuchi01 sparktutorial]$ pwd
/home/spark/sparktutorials/scala/sparktutorial
[spark@dkikuchi01 sparktutorial]$ sbt package
[info] Set current project to Spark Tutorial Project (in build file:/home/spark/sparktutorials/scala/sparktutorial/)
[info] Updating {file:/home/spark/sparktutorials/scala/sparktutorial/}sparktutorial...
[info] Resolving org.fusesource.jansi#jansi;1.4 ...
[info] Done updating.
[info] Compiling 1 Scala source to /home/spark/sparktutorials/scala/sparktutorial/target/scala-2.10/classes...
[info] Packaging /home/spark/sparktutorials/scala/sparktutorial/target/scala-2.10/spark-tutorial-project_2.10-1.0.jar ...
[info] Done packaging.
[success] Total time: 9 s, completed Apr 20, 2015 1:53:35 PM
[spark@dkikuchi01 sparktutorial]$
```

This should compile successfully. Look for the green ‘success’.

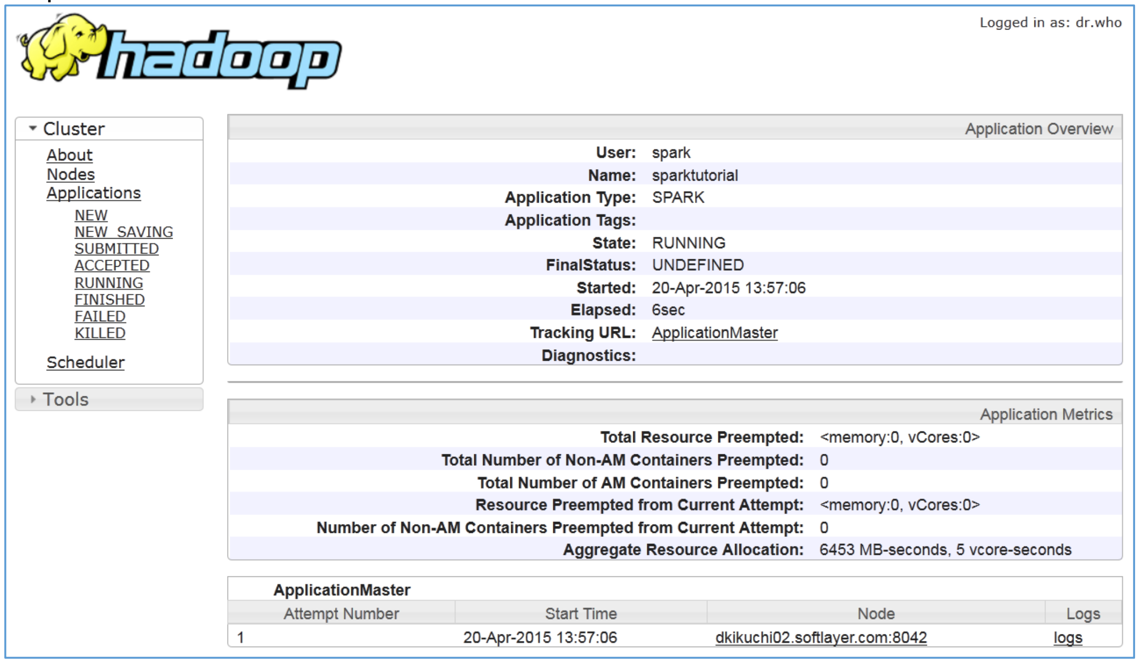
y) Since the compilation and build are successful, the job can be submitted. A shell script was created and available in ‘/home/spark/sparktutorials/scala/sparktutorial/sparktutorial.sh’.
**./sparktutorial.sh**



```
[spark@dkikuchi01 sparktutorial]$ pwd
/home/spark/sparktutorials/scala/sparktutorial
[spark@dkikuchi01 sparktutorial]$ cat sparktutorial.sh
spark-submit --class "sparktutorial" --master yarn-cluster target/scala-2.10/spark-tutorial-project_2.10-1.0.jar
[spark@dkikuchi01 sparktutorial]$
```

‘chmod’ may be required to allow execution of this shell script.

z) The status of job can be viewed while it is running. Go to http://<yourHostName>:8088 where yourHostName is the node where Resource Manager is running. Then click the link where the Name is sparktutorial.



Application Overview

| | |
|---|---|
| User: | spark |
| Name: | sparktutorial |
| Application Type: | SPARK |
| Application Tags: | |
| State: | RUNNING |
| FinalStatus: | UNDEFINED |
| Started: | 20-Apr-2015 13:57:06 |
| Elapsed: | 6sec |
| Tracking URL: | ApplicationMaster |
| Diagnostics: | |

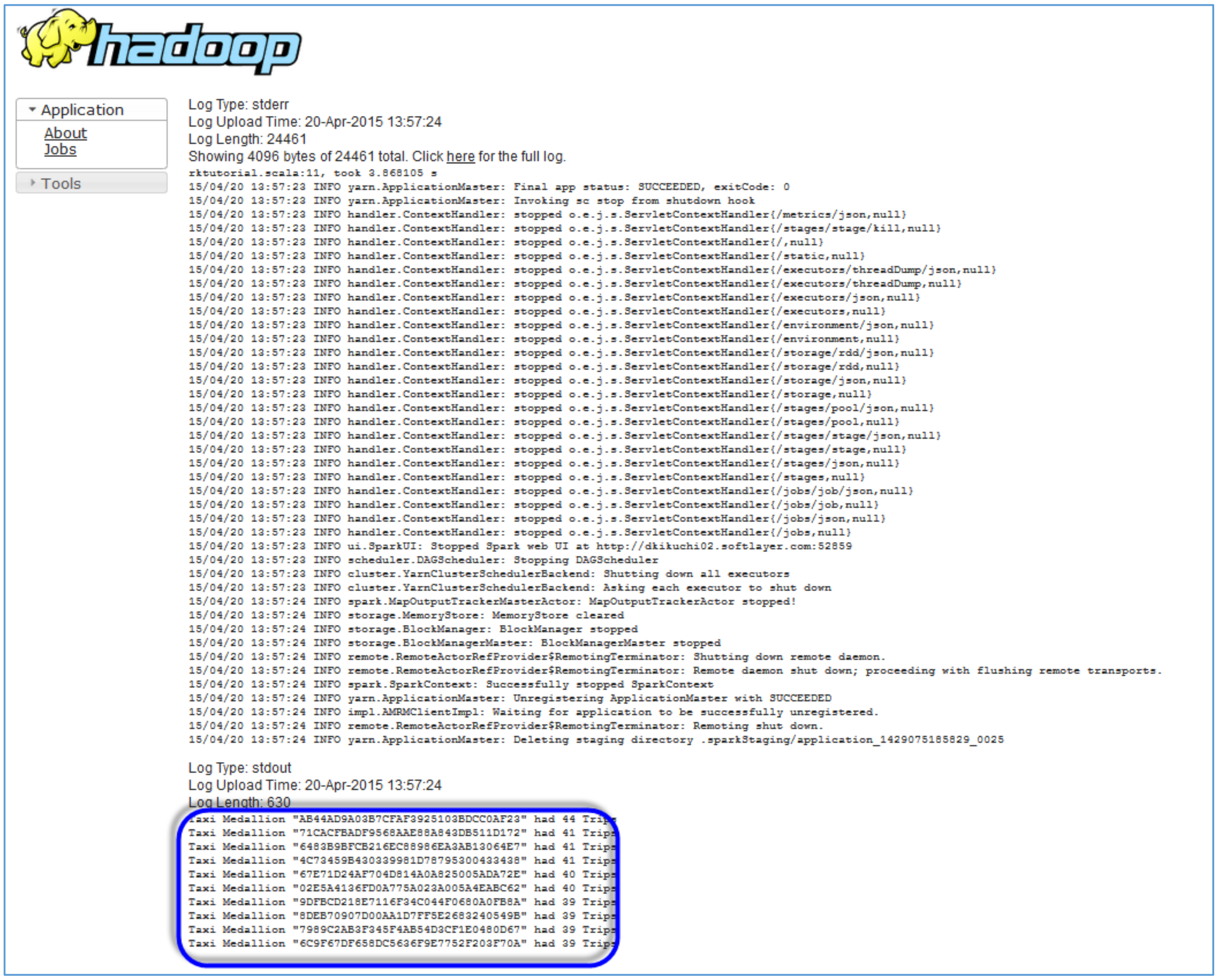
Application Metrics

| | |
|---|---|
| Total Resource Preempted: | <memory:0, vCores:0> |
| Total Number of Non-AM Containers Preempted: | 0 |
| Total Number of AM Containers Preempted: | 0 |
| Resource Preempted from Current Attempt: | <memory:0, vCores:0> |
| Number of Non-AM Containers Preempted from Current Attempt: | 0 |
| Aggregate Resource Allocation: | 6453 MB-seconds, 5 vcore-seconds |

ApplicationMaster

| Attempt Number | Start Time | Node | Logs |
|---|---|---|---|
| 1 | 20-Apr-2015 13:57:06 | dkikuchi02.softlayer.com:8042 | logs |

Once the ‘State:’ is ‘FINISHED’ and the ‘FinalStatus:’ is ‘SUCCEEDED’, click on the ‘logs’ link on the bottom right and scroll down to the bottom. The expected results will be shown.



Log Type: stderr
Log Upload Time: 20-Apr-2015 13:57:24
Log Length: 24461
Showing 4096 bytes of 24461 total. Click here for the full log.

```
rktutorial.scala:11, took 3.868105 s
15/04/20 13:57:23 INFO yarn.ApplicationMaster: Final app status: SUCCEEDED, exitCode: 0
15/04/20 13:57:23 INFO yarn.ApplicationMaster: Invoking sc stop from shutdown hook
15/04/20 13:57:23 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/metrics/json,null}
15/04/20 13:57:23 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages/stage/kill,null}
15/04/20 13:57:23 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/,null}
15/04/20 13:57:23 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/static,null}
15/04/20 13:57:23 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/executors/threadDump/json,null}
15/04/20 13:57:23 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/executors/threadDump,null}
15/04/20 13:57:23 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/executors/json,null}
15/04/20 13:57:23 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/executors,null}
15/04/20 13:57:23 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/environment/json,null}
15/04/20 13:57:23 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/environment,null}
15/04/20 13:57:23 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/storage/rdd/json,null}
15/04/20 13:57:23 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/storage/rdd,null}
15/04/20 13:57:23 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/storage/json,null}
15/04/20 13:57:23 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/storage,null}
15/04/20 13:57:23 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages/pool/json,null}
15/04/20 13:57:23 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages/pool,null}
15/04/20 13:57:23 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages/stage/json,null}
15/04/20 13:57:23 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages/stage,null}
15/04/20 13:57:23 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages/json,null}
15/04/20 13:57:23 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages,null}
15/04/20 13:57:23 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/jobs/job/json,null}
15/04/20 13:57:23 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/jobs/job,null}
15/04/20 13:57:23 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/jobs/json,null}
15/04/20 13:57:23 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/jobs,null}
15/04/20 13:57:23 INFO ui.SparkUI: Stopped Spark web UI at http://dkikuchi02.softlayer.com:52859
15/04/20 13:57:23 INFO scheduler.DAGScheduler: Stopping DAGScheduler
15/04/20 13:57:23 INFO cluster.YarnClusterSchedulerBackend: Shutting down all executors
15/04/20 13:57:23 INFO cluster.YarnClusterSchedulerBackend: Asking each executor to shut down
15/04/20 13:57:24 INFO spark.MapOutputTrackerMasterActor: MapOutputTrackerActor stopped!
15/04/20 13:57:24 INFO storage.MemoryStore: MemoryStore cleared
15/04/20 13:57:24 INFO storage.BlockManager: BlockManager stopped
15/04/20 13:57:24 INFO storage.BlockManagerMaster: BlockManagerMaster stopped
15/04/20 13:57:24 INFO remote.RemoteActorRefProvider$RemotingTerminator: Shutting down remote daemon.
15/04/20 13:57:24 INFO remote.RemoteActorRefProvider$RemotingTerminator: Remote daemon shut down; proceeding with flushing remote transports.
15/04/20 13:57:24 INFO spark.SparkContext: Successfully stopped SparkContext
15/04/20 13:57:24 INFO yarn.ApplicationMaster: Unregistering ApplicationMaster with SUCCEEDED
15/04/20 13:57:24 INFO impl.AMRMClientImpl: Waiting for application to be successfully unregistered.
15/04/20 13:57:24 INFO remote.RemoteActorRefProvider$RemotingTerminator: Remoting shut down.
15/04/20 13:57:24 INFO yarn.ApplicationMaster: Deleting staging directory .sparkStaging/application_1429075185829_0025
```

Log Type: stdout
Log Upload Time: 20-Apr-2015 13:57:24
Log Length: 630

```
Taxi Medallion "AB44AD9A03B7CFAF3925103BDCC0AF23" had 44 Trips
Taxi Medallion "71CACFBADF9568AAE88A843DB511D172" had 41 Trips
Taxi Medallion "6483B9BFCB216EC88986EA3AB13064E7" had 41 Trips
Taxi Medallion "4C73459B430339981D78795300433438" had 41 Trips
Taxi Medallion "67E71D24AF704D814A0A825005ADA72E" had 40 Trips
Taxi Medallion "02E5A4136FD0A775A023A005A4EABC62" had 40 Trips
Taxi Medallion "9DFBCD218E7116F34C044F0680A0FB8A" had 39 Trips
Taxi Medallion "8DEB70907D00AA1D7FF5E2683240549B" had 39 Trips
Taxi Medallion "7989C2AB3F345F4AB54D3CF1E0480D67" had 39 Trips
Taxi Medallion "6C9F67DF658DC5636F9E7752F203F70A" had 39 Trips
```

## Congratulations, you have complete this lab!

Great work and congratulations, you have completed this lab.

Screen captures in this document may vary slightly from yours.

# Lab 2 Getting Started with Spark MLlib

This lab focuses on MLlib, Spark's scalable machine learning library. The K-Means algorithm will be applied to the drop-off latitudes and longitudes of taxis for 3 clusters. As such, it may give a good indication of an area where to best hail a cab. This will be implemented in the Spark shell as well as a Spark application in Scala. BigInsights Version 4 was used to create this tutorial and provides Spark 1.2.1 and Scala 2.10.4. The 'sbt' source build tool 0.13.7 was used to build the Spark application in this tutorial. Ant or Maven are alternative options.

Hope this proves to be a helpful tutorial as an introduction to MLlib, Spark and Scala.

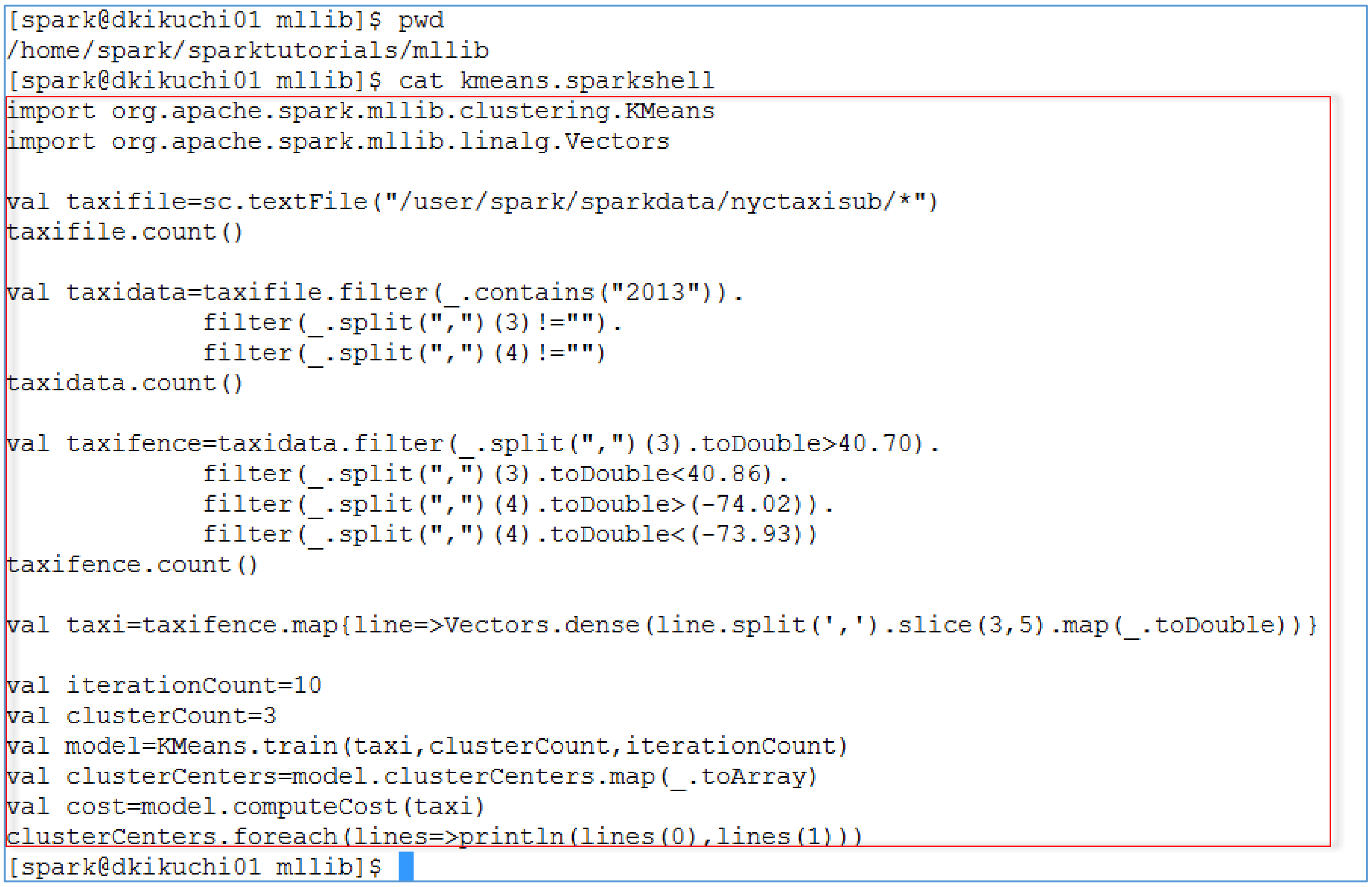
## Part 1 Initial Set Up with Sample Data and Code

If not already completed, please go through the steps in Lab 1, Part 1.

## Part 2 Using the Spark Shell

In this section, the Spark shell will be used to acquire the desired results, i.e. K-Means clustering for drop-off latitudes and longitudes of taxis for 3 clusters. The sample data contains a subset of taxi trips with hack license, medallion, pickup date/time, drop-off date/time, pickup latitude/longitude, drop-off latitude/longitude, passenger count, trip distance, trip time and other information.

The bold commands from this document can be copied and pasted to the Spark shell, and they are also available in '/home/spark/sparktutorials/mllib/kmeans.sparkshell'.



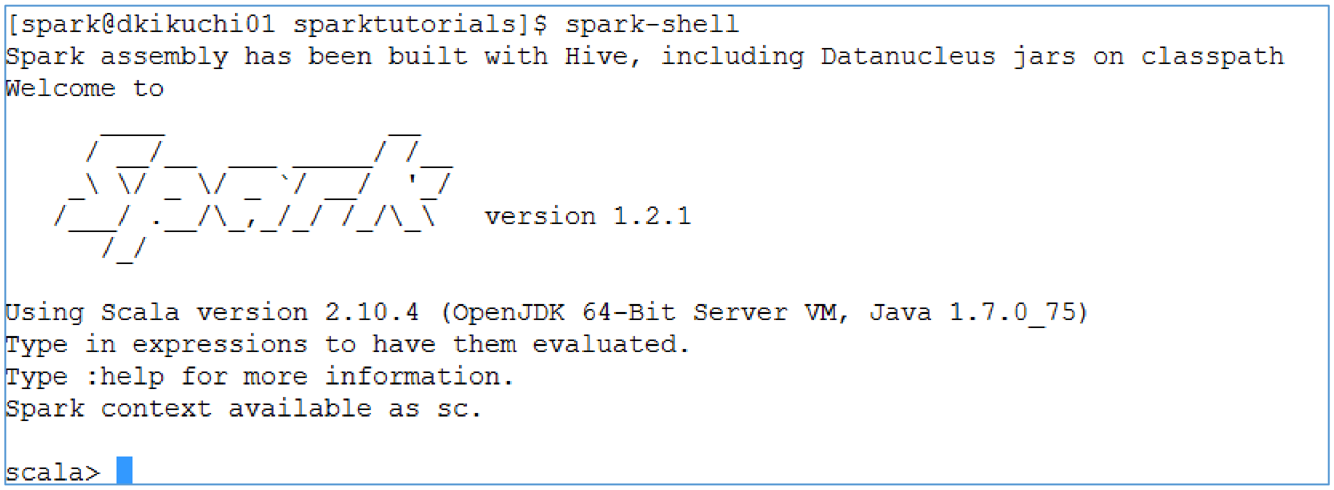
```
[spark@dkikuchi01 mllib]$ pwd
/home/spark/sparktutorials/mllib
[spark@dkikuchi01 mllib]$ cat kmeans.sparkshell
import org.apache.spark.mllib.clustering.KMeans
import org.apache.spark.mllib.linalg.Vectors

val taxifile=sc.textFile("/user/spark/sparkdata/nyctaxisub/*")
taxifile.count()

val taxidata=taxifile.filter(_.contains("2013")).
             filter(_.split(",")(3)!="").
             filter(_.split(",")(4)!="")
taxidata.count()

val taxifence=taxidata.filter(_.split(",")(3).toDouble>40.70).
             filter(_.split(",")(3).toDouble<40.86).
             filter(_.split(",")(4).toDouble>(-74.02)).
             filter(_.split(",")(4).toDouble<(-73.93))
taxifence.count()

val taxi=taxifence.map{line=>Vectors.dense(line.split(',').slice(3,5).map(_.toDouble))}

val iterationCount=10
val clusterCount=3
val model=KMeans.train(taxi,clusterCount,iterationCount)
val clusterCenters=model.clusterCenters.map(_.toArray)
val cost=model.computeCost(taxi)
clusterCenters.foreach(lines=>println(lines(0),lines(1)))
[spark@dkikuchi01 mllib]$
```

The inner red box shows the contents of this file.

The Spark shell can be verbose. To view ERRORs only, a 'log4j.properties' file has been provided in '/home/spark/sparktutorials'. Please launch the 'spark-shell' in this subdirectory to suppress the verbose output.

a) Please ensure that the BigInsights Version 4 environment has been started. Login as userid 'spark' or alternatively, login in as 'root' then 'su - spark'.

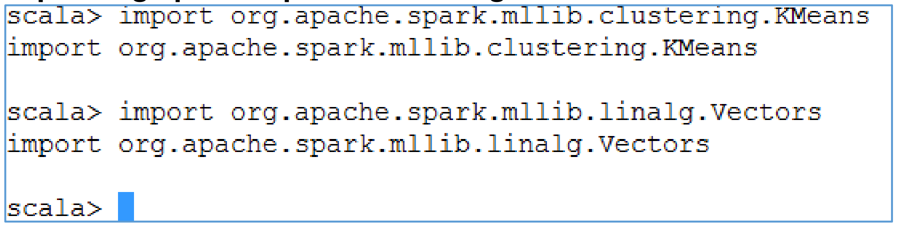
b) Fire up the Spark shell from '/home/spark/sparktutorials'
**spark-shell**



```
[spark@dkikuchi01 sparktutorials]$ spark-shell
Spark assembly has been built with Hive, including Datanucleus jars on classpath
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.2.1
      /_/

Using Scala version 2.10.4 (OpenJDK 64-Bit Server VM, Java 1.7.0_75)
Type in expressions to have them evaluated.
Type :help for more information.
Spark context available as sc.

scala>
```

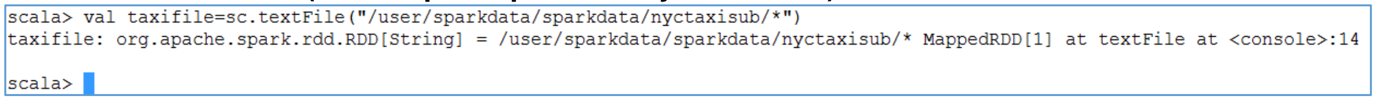
Note that this version is Spark 1.2.1 using Scala version 2.10.4 on OpenJDK 64-Bit Server VM, Java 1.7.0_75).

c) Import needed packages for K-Means algorithm and Vectors packages.
**import org.apache.spark.mllib.clustering.KMeans**
**import org.apache.spark.mllib.linalg.Vectors**



```
scala> import org.apache.spark.mllib.clustering.KMeans
import org.apache.spark.mllib.clustering.KMeans

scala> import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.linalg.Vectors

scala>
```

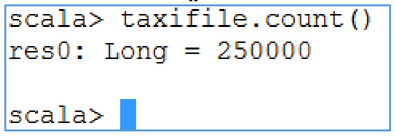
d) Create an RDD from the HDFS data in '/user/spark/sparkdata/nyctaxisub/nyctaxisub.csv'.
**val taxifile=sc.textFile("/user/spark/sparkdata/nyctaxisub/*")**



```
scala> val taxifile=sc.textFile("/user/sparkdata/sparkdata/nyctaxisub/*")
taxifile: org.apache.spark.rdd.RDD[String] = /user/sparkdata/sparkdata/nyctaxisub/* MappedRDD[1] at textFile at <console>:14

scala>
```

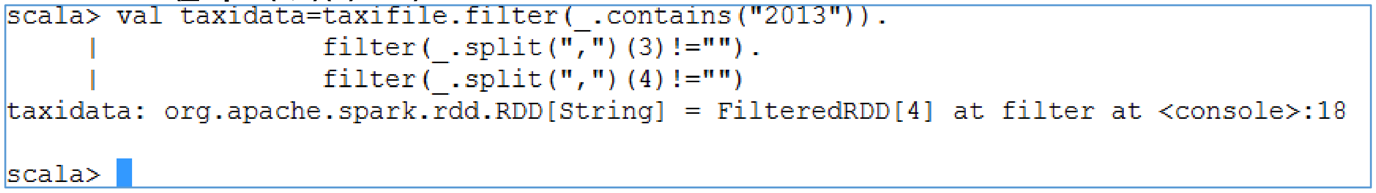
Keep in mind that RDDs are not processed until an action is performed.

e) Determine the number of rows in 'taxifile'.
**taxifile.count()**



```
scala> taxifile.count()
res0: Long = 250000

scala>
```

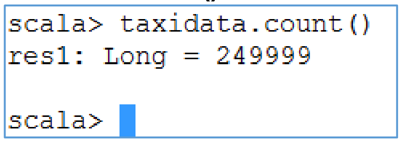
f) To prepare and cleanse the data, issue the following.
**val taxidata=taxifile.filter(_.contains("2013")).**
**filter(_.split(",")(3)!="").**
**filter(_.split(",")(4)!="")**



```
scala> val taxidata=taxifile.filter(_.contains("2013")).
     |                  filter(_.split(",")(3)!="").
     |                  filter(_.split(",")(4)!="")
taxidata: org.apache.spark.rdd.RDD[String] = FilteredRDD[4] at filter at <console>:18

scala>
```

The first filter limits the rows to those that occurred in the year 2013. This will also remove header(s). The third and fourth column contain the drop-off latitude and longitude. The algorithm will throw exceptions if these values are empty.

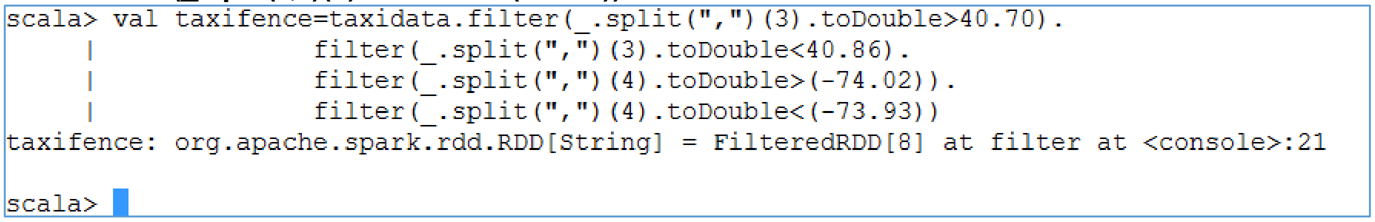
g) Determine the number of rows in ‘taxidata’.
**taxidata.count()**



```
scala> taxidata.count()
res1: Long = 249999

scala>
```

In this case, only the header was removed.

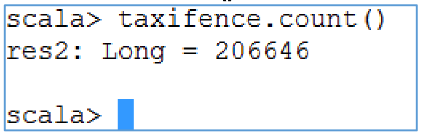
h) To fence the area roughly to New York City, please issue the following.
**val taxifence=taxidata.filter(_.split(",")(3).toDouble>40.70).**
**filter(_.split(",")(3).toDouble<40.86).**
**filter(_.split(",")(4).toDouble>(-74.02)).**
**filter(_.split(",")(4).toDouble<(-73.93))**



```
scala> val taxifence=taxidata.filter(_.split(",")(3).toDouble>40.70).
     |              filter(_.split(",")(3).toDouble<40.86).
     |              filter(_.split(",")(4).toDouble>(-74.02)).
     |              filter(_.split(",")(4).toDouble<(-73.93))
taxifence: org.apache.spark.rdd.RDD[String] = FilteredRDD[8] at filter at <console>:21

scala>
```

The data contains many drop-off points outside of New York City. To get more accurate clusters, drop-off latitudes and longitudes are fenced in the given values.

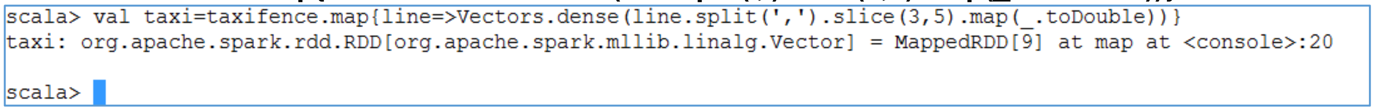
i) Determine the number of rows in ‘taxifence’.
**taxifence.count()**



```
scala> taxifence.count()
res2: Long = 206646

scala>
```

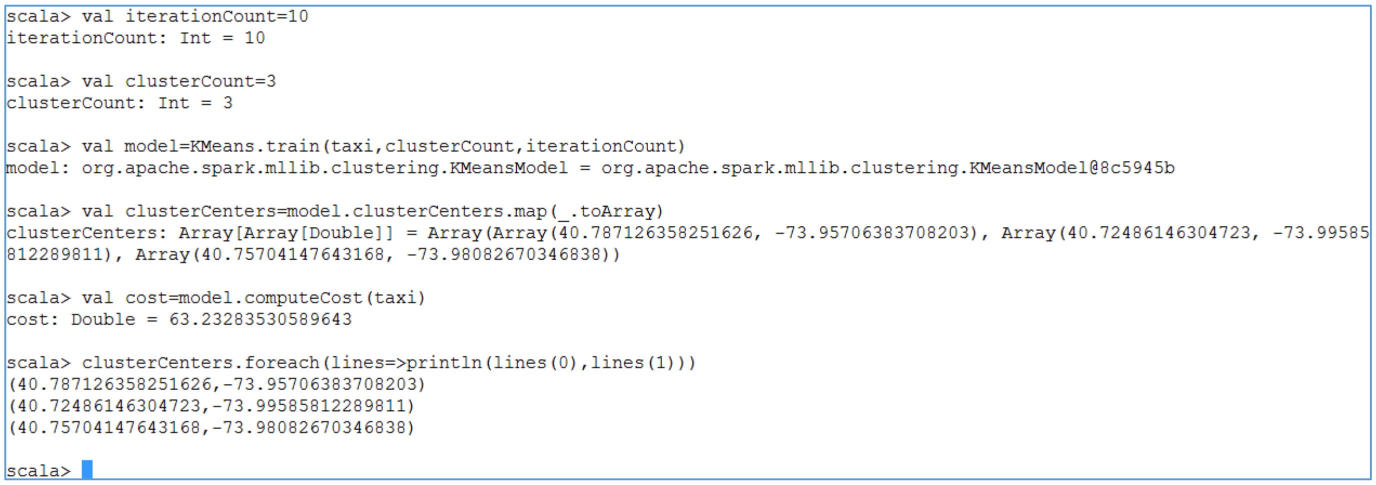
Approximately, 43,353 rows were dropped since these drop-off points are outside of New York City.

j) Create Vectors with the latitudes and longitudes that will be used as input to the K-Means algorithm.
**val taxi=taxifence.map{line=>Vectors.dense(line.split(',').slice(3,5).map(_.toDouble))}**



```
scala> val taxi=taxifence.map{line=>Vectors.dense(line.split(',').slice(3,5).map(_.toDouble))}
taxi: org.apache.spark.rdd.RDD[org.apache.spark.mllib.linalg.Vector] = MappedRDD[9] at map at <console>:20

scala>
```

k) Run the K-Means algorithm.
**val iterationCount=10**
**val clusterCount=3**
**val model=KMeans.train(taxi,clusterCount,iterationCount)**
**val clusterCenters=model.clusterCenters.map(_.toArray)**
**val cost=model.computeCost(taxi)**
**clusterCenters.foreach(lines=>println(lines(0),lines(1)))**



```
scala> val iterationCount=10
iterationCount: Int = 10

scala> val clusterCount=3
clusterCount: Int = 3

scala> val model=KMeans.train(taxi,clusterCount,iterationCount)
model: org.apache.spark.mllib.clustering.KMeansModel = org.apache.spark.mllib.clustering.KMeansModel@8c5945b

scala> val clusterCenters=model.clusterCenters.map(_.toArray)
clusterCenters: Array[Array[Double]] = Array(Array(40.787126358251626, -73.95706383708203), Array(40.72486146304723, -73.99585
812289811), Array(40.75704147643168, -73.98082670346838))

scala> val cost=model.computeCost(taxi)
cost: Double = 63.23283530589643

scala> clusterCenters.foreach(lines=>println(lines(0),lines(1)))
(40.787126358251626,-73.95706383708203)
(40.72486146304723,-73.99585812289811)
(40.75704147643168,-73.98082670346838)

scala> 
```

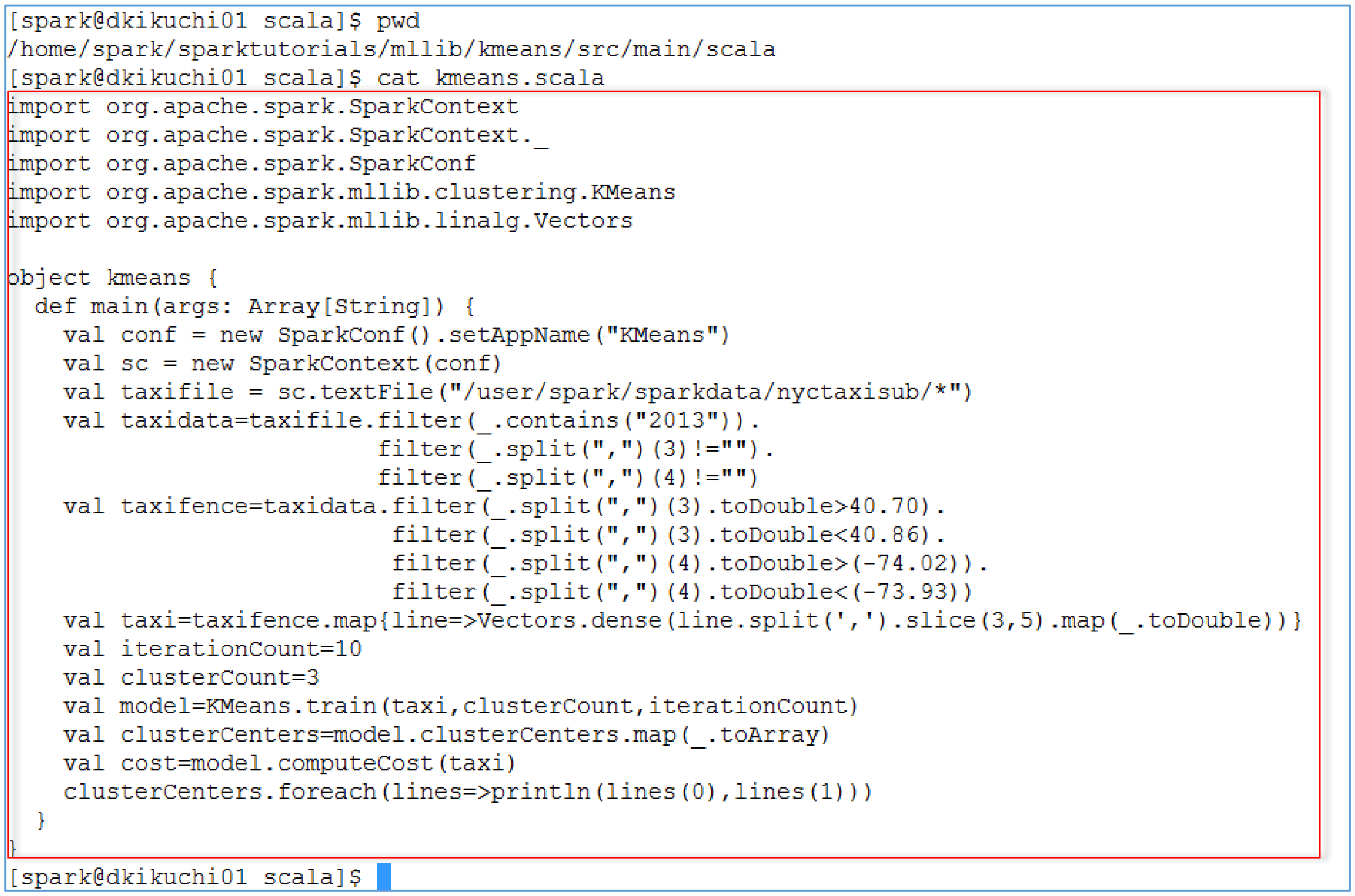
This may take a few minutes so please wait.  Not surprisingly, the first point is the Upper East Side, presumably where people are more likely to take cabs than subways.  The second point is in The Village, NYU, Soho and Little Italy area.  And the third point is between the Theater District and Grand Central.

l) Exit the Spark shell by typing **exit** and enter.

## Part 3 Creating a Spark Application

In this section, a Spark application will be created to acquire the desired results.

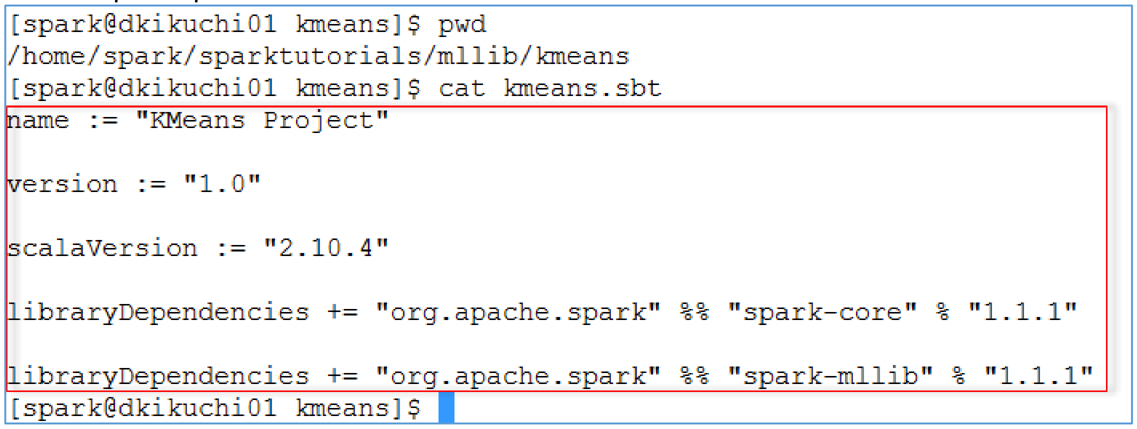
m) Please go to '/home/spark/sparktutorials/mllib/kmeans/src/main/scala' and edit the 'kmeans.scala' file.



```
[spark@dkikuchi01 scala]$ pwd
/home/spark/sparktutorials/mllib/kmeans/src/main/scala
[spark@dkikuchi01 scala]$ cat kmeans.scala
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
import org.apache.spark.mllib.clustering.KMeans
import org.apache.spark.mllib.linalg.Vectors

object kmeans {
  def main(args: Array[String]) {
    val conf = new SparkConf().setAppName("KMeans")
    val sc = new SparkContext(conf)
    val taxifile = sc.textFile("/user/spark/sparkdata/nyctaxisub/*")
    val taxidata=taxifile.filter(_.contains("2013")).
                          filter(_.split(",")(3)!="").
                          filter(_.split(",")(4)!="")
    val taxifence=taxidata.filter(_.split(",")(3).toDouble>40.70).
                           filter(_.split(",")(3).toDouble<40.86).
                           filter(_.split(",")(4).toDouble>(-74.02)).
                           filter(_.split(",")(4).toDouble<(-73.93))
    val taxi=taxifence.map{line=>Vectors.dense(line.split(',').slice(3,5).map(_.toDouble))}
    val iterationCount=10
    val clusterCount=3
    val model=KMeans.train(taxi,clusterCount,iterationCount)
    val clusterCenters=model.clusterCenters.map(_.toArray)
    val cost=model.computeCost(taxi)
    clusterCenters.foreach(lines=>println(lines(0),lines(1)))
  }
}
[spark@dkikuchi01 scala]$
```

The actual code is in the inner red box. Package imports are performed. In the ‘main’, the SparkContext is created and assigned to ‘sc’. The code following this should look familiar as these are the commands issued in the previous steps for the Spark shell implementation. The results will be written to the log in this case, but it could also be written to HDFS.

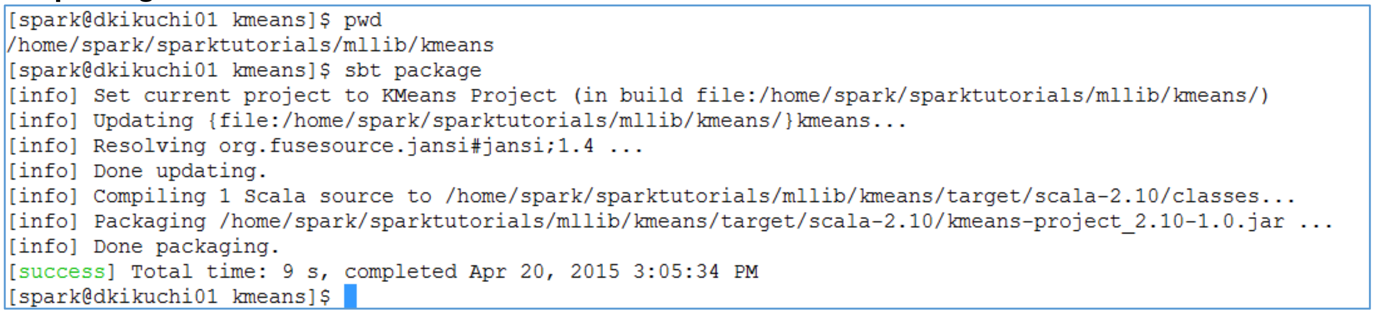
n) ‘sbt’ 0.13.7 was installed and used for the build, however Maven or Ant could be used. Go to ‘/home/spark/sparktutorials/mllib/kmeans’ and view ‘kmeans.sbt’.



```
[spark@dkikuchi01 kmeans]$ pwd
/home/spark/sparktutorials/mllib/kmeans
[spark@dkikuchi01 kmeans]$ cat kmeans.sbt
name := "KMeans Project"

version := "1.0"

scalaVersion := "2.10.4"

libraryDependencies += "org.apache.spark" %% "spark-core" % "1.1.1"

libraryDependencies += "org.apache.spark" %% "spark-mllib" % "1.1.1"
[spark@dkikuchi01 kmeans]$
```

This provides the name, versions and library dependencies for the build tool.

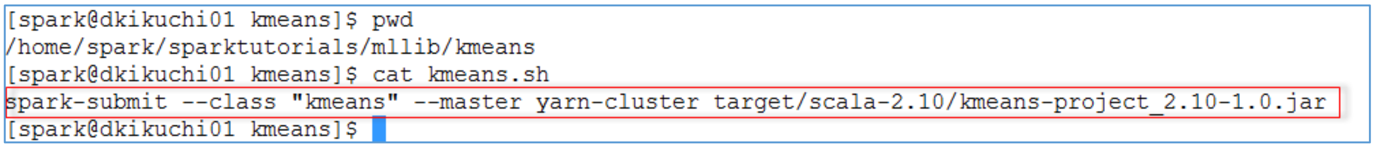
o) (OPTIONAL) In order to compile this source, 'sbt' will need to be downloaded and installed. The build tool is licensed and located at 'http://www.scala-sbt.org'. However, the source file has been pre-compiled and available. If you do not wish to install 'sbt', please go on to the next step. Otherwise, to re-compile the code, download and install 'sbt', go to '/home/spark/sparktutorials/scala/sparktutorial' and issue.
**sbt package**



```
[spark@dkikuchi01 kmeans]$ pwd
/home/spark/sparktutorials/mllib/kmeans
[spark@dkikuchi01 kmeans]$ sbt package
[info] Set current project to KMeans Project (in build file:/home/spark/sparktutorials/mllib/kmeans/)
[info] Updating {file:/home/spark/sparktutorials/mllib/kmeans/}kmeans...
[info] Resolving org.fusesource.jansi#jansi;1.4 ...
[info] Done updating.
[info] Compiling 1 Scala source to /home/spark/sparktutorials/mllib/kmeans/target/scala-2.10/classes...
[info] Packaging /home/spark/sparktutorials/mllib/kmeans/target/scala-2.10/kmeans-project_2.10-1.0.jar ...
[info] Done packaging.
[success] Total time: 9 s, completed Apr 20, 2015 3:05:34 PM
[spark@dkikuchi01 kmeans]$
```

This should compile successfully. Look for the green 'success'.

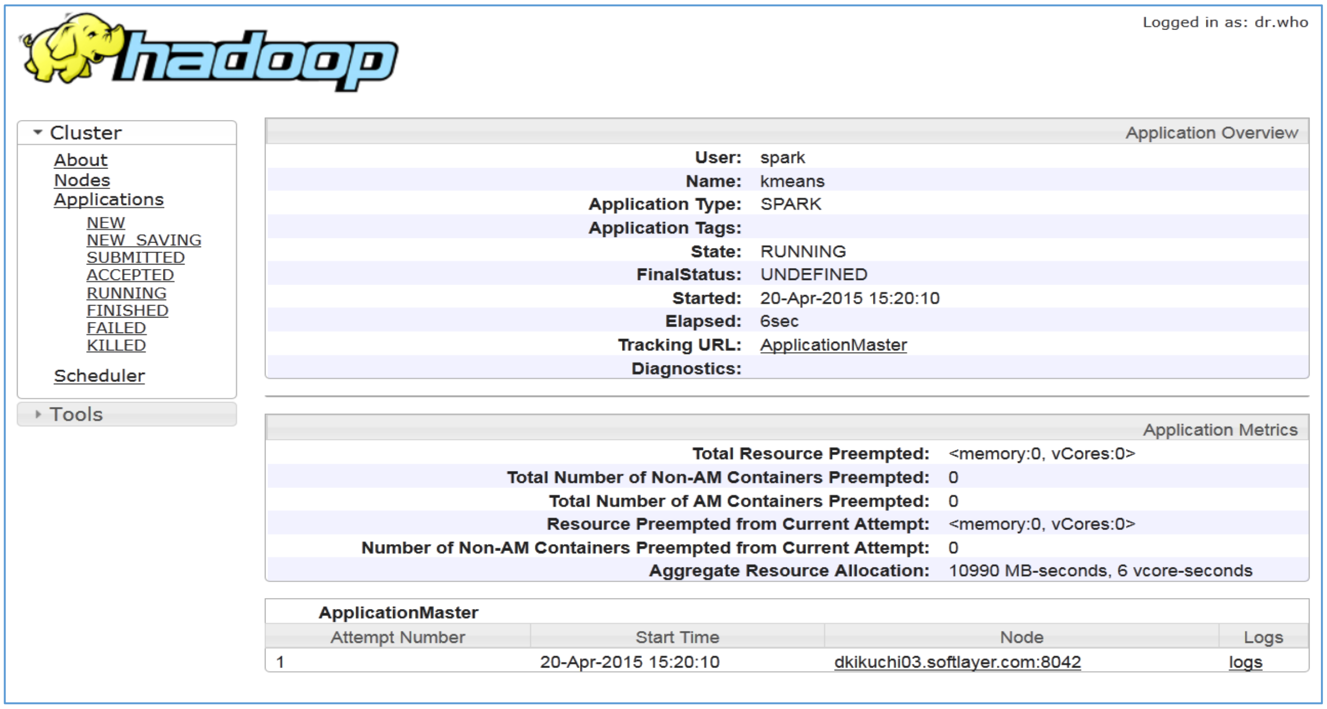
p) Since the compilation and build are successful, the job can be submitted. A shell script was created and available in '/home/spark/sparktutorials/mllib/kmeans/kmeans.sh'.
**./kmeans.sh**



```
[spark@dkikuchi01 kmeans]$ pwd
/home/spark/sparktutorials/mllib/kmeans
[spark@dkikuchi01 kmeans]$ cat kmeans.sh
spark-submit --class "kmeans" --master yarn-cluster target/scala-2.10/kmeans-project_2.10-1.0.jar
[spark@dkikuchi01 kmeans]$
```

'chmod' may be required to allow execution of this shell script.

q) The status of job can be viewed while it is running. Go to http://<yourHostName>:8088 where yourHostName is the node where Resource Manager is running. Then click the link where the Name is kmeans.



Application Overview

| | |
|---|---|
| User: | spark |
| Name: | kmeans |
| Application Type: | SPARK |
| Application Tags: | |
| State: | RUNNING |
| FinalStatus: | UNDEFINED |
| Started: | 20-Apr-2015 15:20:10 |
| Elapsed: | 6sec |
| Tracking URL: | ApplicationMaster |
| Diagnostics: | |

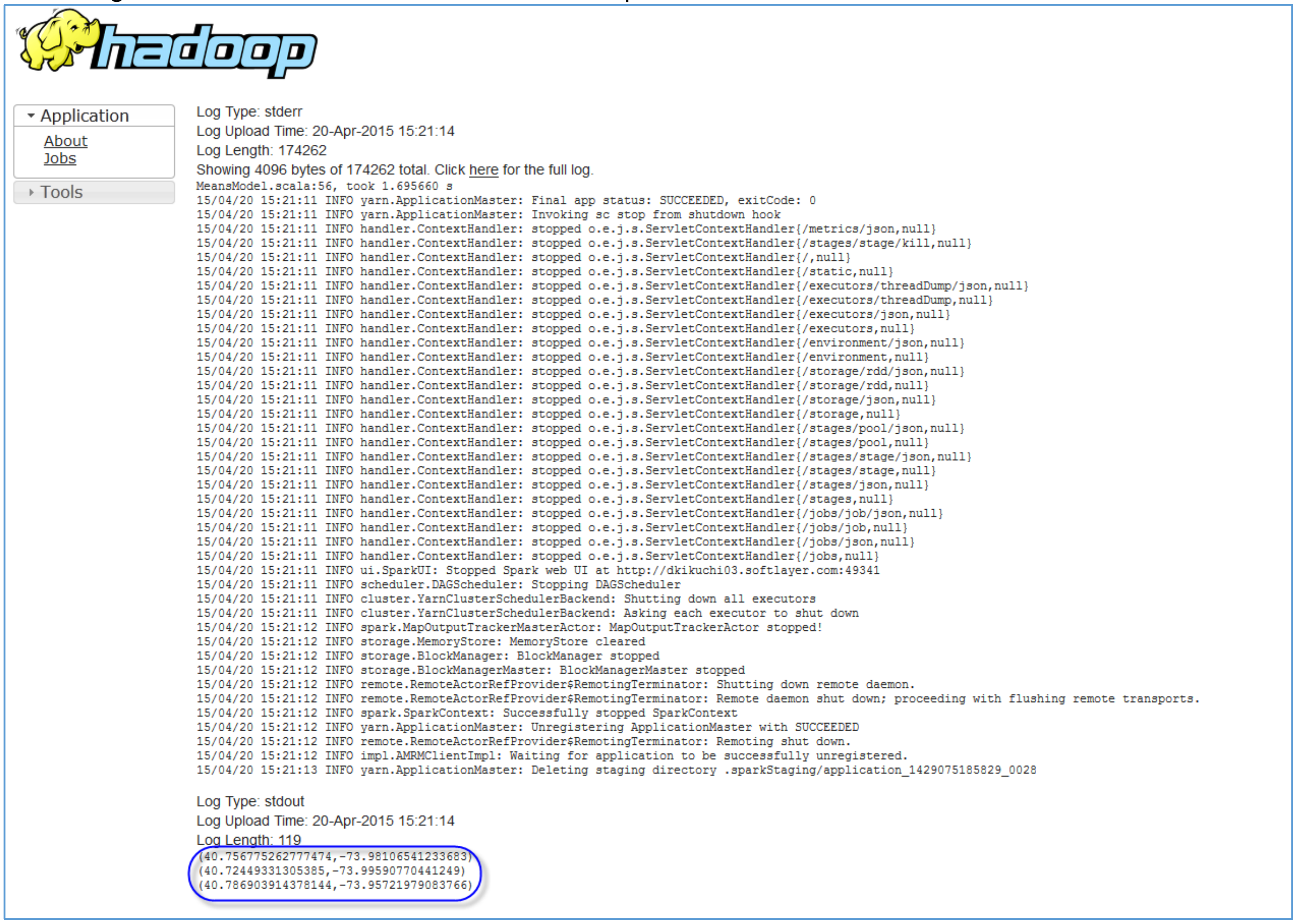
Application Metrics

| | |
|---|---|
| Total Resource Preempted: | <memory:0, vCores:0> |
| Total Number of Non-AM Containers Preempted: | 0 |
| Total Number of AM Containers Preempted: | 0 |
| Resource Preempted from Current Attempt: | <memory:0, vCores:0> |
| Number of Non-AM Containers Preempted from Current Attempt: | 0 |
| Aggregate Resource Allocation: | 10990 MB-seconds, 6 vcore-seconds |

ApplicationMaster

| Attempt Number | Start Time | Node | Logs |
|---|---|---|---|
| 1 | 20-Apr-2015 15:20:10 | dkikuchi03.softlayer.com:8042 | logs |

Once the ‘State:’ is ‘FINISHED’ and the ‘FinalStatus:’ is ‘SUCCEEDED’, click on the ‘logs’ link on the bottom right and scroll down to the bottom. The expected results will be shown.



Log Type: stderr
Log Upload Time: 20-Apr-2015 15:21:14
Log Length: 174262
Showing 4096 bytes of 174262 total. Click here for the full log.

```
MeansModel.scala:56, took 1.695660 s
15/04/20 15:21:11 INFO yarn.ApplicationMaster: Final app status: SUCCEEDED, exitCode: 0
15/04/20 15:21:11 INFO yarn.ApplicationMaster: Invoking sc stop from shutdown hook
15/04/20 15:21:11 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/metrics/json,null}
15/04/20 15:21:11 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages/stage/kill,null}
15/04/20 15:21:11 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/,null}
15/04/20 15:21:11 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/static,null}
15/04/20 15:21:11 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/executors/threadDump/json,null}
15/04/20 15:21:11 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/executors/threadDump,null}
15/04/20 15:21:11 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/executors/json,null}
15/04/20 15:21:11 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/executors,null}
15/04/20 15:21:11 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/environment/json,null}
15/04/20 15:21:11 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/environment,null}
15/04/20 15:21:11 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/storage/rdd/json,null}
15/04/20 15:21:11 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/storage/rdd,null}
15/04/20 15:21:11 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/storage/json,null}
15/04/20 15:21:11 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/storage,null}
15/04/20 15:21:11 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages/pool/json,null}
15/04/20 15:21:11 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages/pool,null}
15/04/20 15:21:11 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages/stage/json,null}
15/04/20 15:21:11 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages/stage,null}
15/04/20 15:21:11 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages/json,null}
15/04/20 15:21:11 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages,null}
15/04/20 15:21:11 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/jobs/job/json,null}
15/04/20 15:21:11 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/jobs/job,null}
15/04/20 15:21:11 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/jobs/json,null}
15/04/20 15:21:11 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/jobs,null}
15/04/20 15:21:11 INFO ui.SparkUI: Stopped Spark web UI at http://dkikuchi03.softlayer.com:49341
15/04/20 15:21:11 INFO scheduler.DAGScheduler: Stopping DAGScheduler
15/04/20 15:21:11 INFO cluster.YarnClusterSchedulerBackend: Shutting down all executors
15/04/20 15:21:11 INFO cluster.YarnClusterSchedulerBackend: Asking each executor to shut down
15/04/20 15:21:12 INFO spark.MapOutputTrackerMasterActor: MapOutputTrackerActor stopped!
15/04/20 15:21:12 INFO storage.MemoryStore: MemoryStore cleared
15/04/20 15:21:12 INFO storage.BlockManager: BlockManager stopped
15/04/20 15:21:12 INFO storage.BlockManagerMaster: BlockManagerMaster stopped
15/04/20 15:21:12 INFO remote.RemoteActorRefProvider$RemotingTerminator: Shutting down remote daemon.
15/04/20 15:21:12 INFO remote.RemoteActorRefProvider$RemotingTerminator: Remote daemon shut down; proceeding with flushing remote transports.
15/04/20 15:21:12 INFO spark.SparkContext: Successfully stopped SparkContext
15/04/20 15:21:12 INFO yarn.ApplicationMaster: Unregistering ApplicationMaster with SUCCEEDED
15/04/20 15:21:12 INFO remote.RemoteActorRefProvider$RemotingTerminator: Remoting shut down.
15/04/20 15:21:12 INFO impl.AMRMClientImpl: Waiting for application to be successfully unregistered.
15/04/20 15:21:13 INFO yarn.ApplicationMaster: Deleting staging directory .sparkStaging/application_1429075185829_0028
```

Log Type: stdout
Log Upload Time: 20-Apr-2015 15:21:14
Log Length: 119

```
(40.756775262777474,-73.98106541233683)
(40.72449331305385,-73.99590770441249)
(40.786903914378144,-73.95721979083766)
```

r) Congratulations, you have complete this lab!

Great work and congratulations, you have completed this lab.

Screen captures in this document may vary slightly from yours.

# Lab 3 Getting Started with Spark Streaming

This lab focuses on Spark Streaming, an easy to build, scalable, stateful (e.g. sliding windows) stream processing library. Streaming jobs are written in a similar way that Spark batch jobs are. In this exercise, taxi trip data will be streamed using a socket connection and then analyzed to provide a summary of number of passengers by taxi vendor. This will be implemented in the Spark shell using Scala. A Spark application could also be created using this code. Please refer to the previously labs for guidance. A standalone Python script is included that will take the taxi trip data file and send rows when a socket connection from Spark Streaming is made. BigInsights Version 4 was used and provides Spark 1.2.1 and Scala 2.10.4.

We hope this proves to be a helpful tutorial as an introduction to Spark Streaming, Spark and Scala.

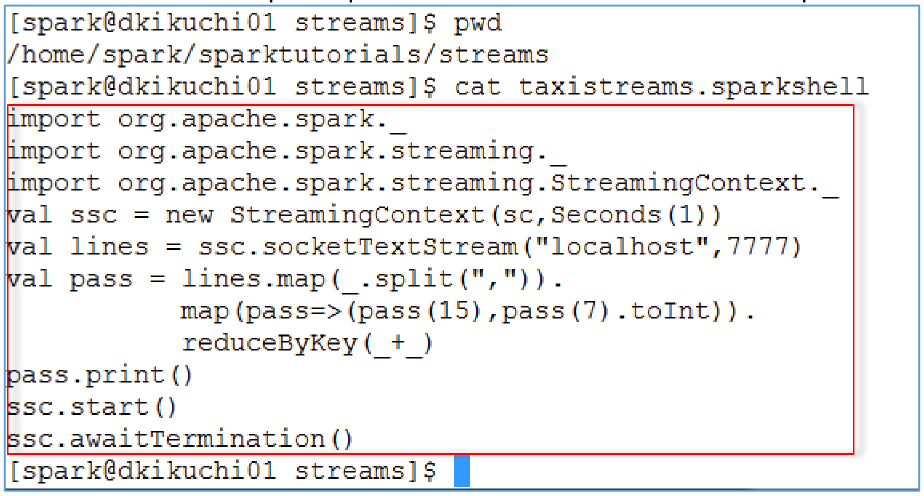
## Part 1 Initial Set Up with Sample Data and Code

If not already complete, please go through the steps in Lab 1, Part 1.

## Part 2 Using the Spark Shell

In this section, the Spark shell will be used to acquire the desired results, i.e. a running summary of number of passengers by taxi vendor. The sample file contains taxi trip data with hack license, medallion, pickup date/time, drop off date/time, pickup latitude/longitude, drop off latitude/longitude, passenger count, trip distance, trip time, vendor and other information.

The bold commands from this document can be copied and pasted to the Spark shell, and they are also available in '/home/spark/sparktutorials/streams/taxistreams.sparkshell'.



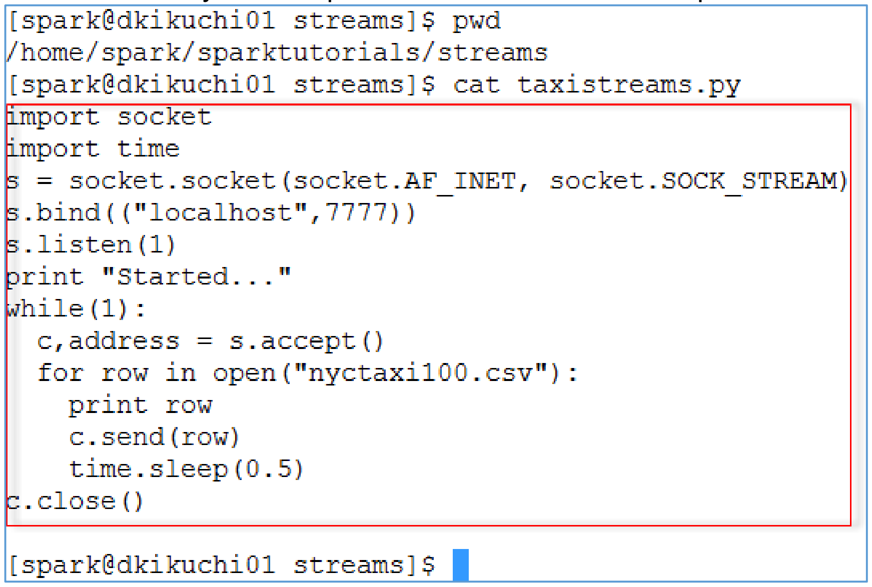
```
[spark@dkikuchi01 streams]$ pwd
/home/spark/sparktutorials/streams
[spark@dkikuchi01 streams]$ cat taxistreams.sparkshell
import org.apache.spark._
import org.apache.spark.streaming._
import org.apache.spark.streaming.StreamingContext._
val ssc = new StreamingContext(sc,Seconds(1))
val lines = ssc.socketTextStream("localhost",7777)
val pass = lines.map(_.split(",")).
           map(pass=>(pass(15),pass(7).toInt)).
           reduceByKey(_+_)
pass.print()
ssc.start()
ssc.awaitTermination()
[spark@dkikuchi01 streams]$
```

The inner red box shows the contents of this file.

The Spark shell can be verbose. To view ERRORs only, a 'log4j.properties' file has been provided in /home/spark/sparktutorials. Please launch the 'spark-shell' in this subdirectory to suppress the verbose output.

a) Please ensure that the BigInsights Version 4 environment has been started. Login as userid ‘spark’ or alternatively login as ‘root’ and then ‘su - spark’.

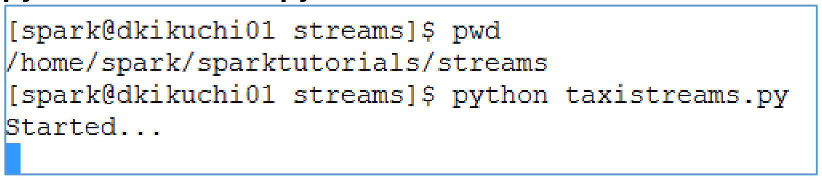
b) A standalone Python script was written to send taxi trip data rows to a socket connection.



```
[spark@dkikuchi01 streams]$ pwd
/home/spark/sparktutorials/streams
[spark@dkikuchi01 streams]$ cat taxistreams.py
import socket
import time
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(("localhost",7777))
s.listen(1)
print "Started..."
while(1):
  c,address = s.accept()
  for row in open("nyctaxi100.csv"):
    print row
    c.send(row)
    time.sleep(0.5)
c.close()

[spark@dkikuchi01 streams]$
```

Once started, the program will bind and listen to the localhost socket 7777. When a connection is made, it will read ‘nyctaxi100.csv’ and send across the socket. The sleep is set such that one line will be sent every 0.5 seconds, or 2 rows a second. This was intentionally set to a high value to make it easier to view the data during execution.

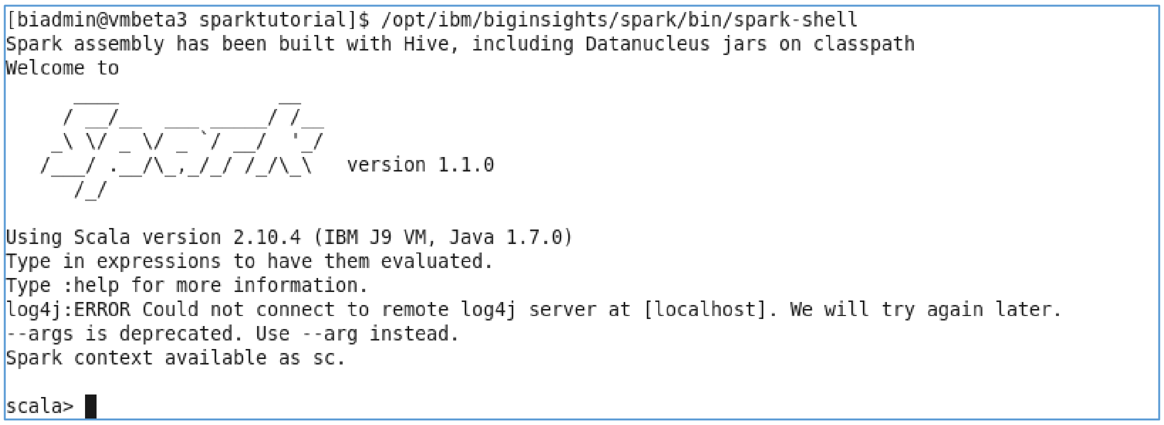
c) To invoke the standalone Python program, issue the following command in the ‘/home/spark/sparktutorials/streams’ folder.
**python taxistreams.py**



```
[spark@dkikuchi01 streams]$ pwd
/home/spark/sparktutorials/streams
[spark@dkikuchi01 streams]$ python taxistreams.py
Started...
```

The program has been started and is awaiting Spark Streaming to connect and receive the data.

d) Open another terminal and fire up the Spark shell from ‘/home/spark/sparktutorials’
**spark-shell**

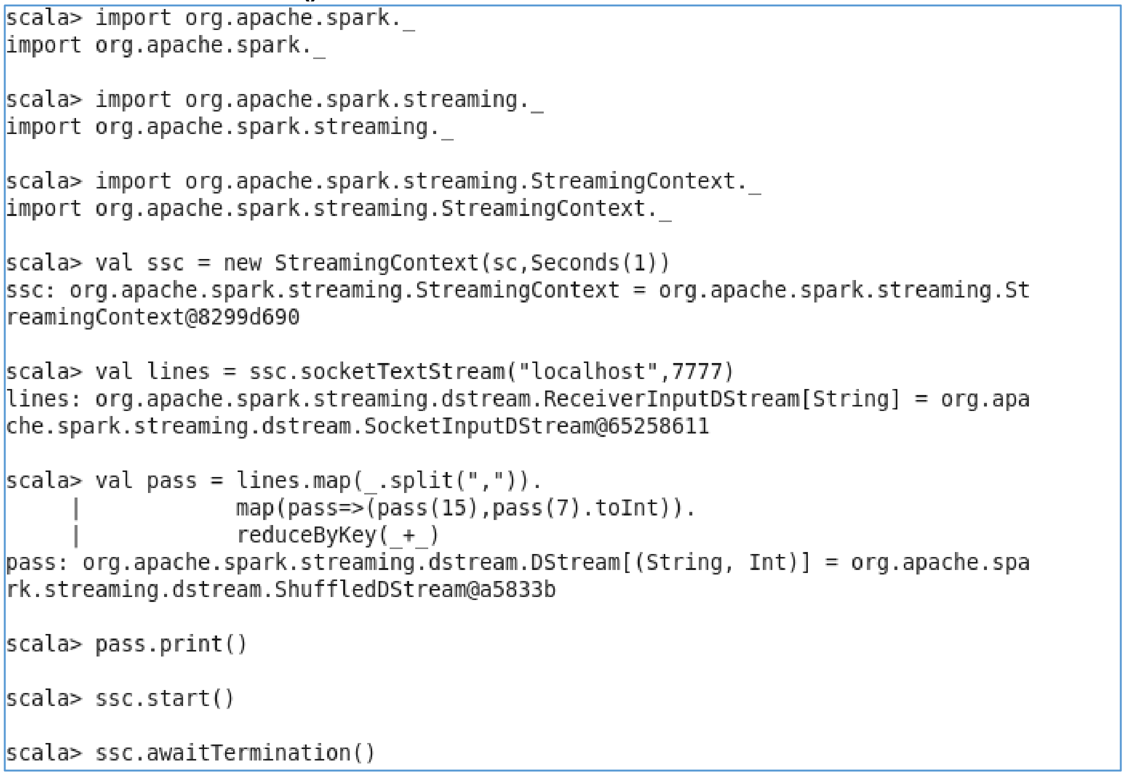


```
[biadmin@vmbeta3 sparktutorial]$ /opt/ibm/biginsights/spark/bin/spark-shell
Spark assembly has been built with Hive, including Datanucleus jars on classpath
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.1.0
      /_/

Using Scala version 2.10.4 (IBM J9 VM, Java 1.7.0)
Type in expressions to have them evaluated.
Type :help for more information.
log4j:ERROR Could not connect to remote log4j server at [localhost]. We will try again later.
--args is deprecated. Use --arg instead.
Spark context available as sc.

scala> 
```

Note that this version is Spark 1.2.1 using Scala version 2.10.4 on OpenJDK 64-Bit Server VM, Java 1.7.0_75).

e) Copy and paste the following code into the spark-shell.

```
import org.apache.spark._
import org.apache.spark.streaming._
import org.apache.spark.streaming.StreamingContext._
val ssc = new StreamingContext(sc,Seconds(1))
val lines = ssc.socketTextStream("localhost",7777)
val pass = lines.map(_.split(",")).
        map(pass=>(pass(15),pass(7).toInt)).
        reduceByKey(_+_)
pass.print()
ssc.start()
ssc.awaitTermination()
```



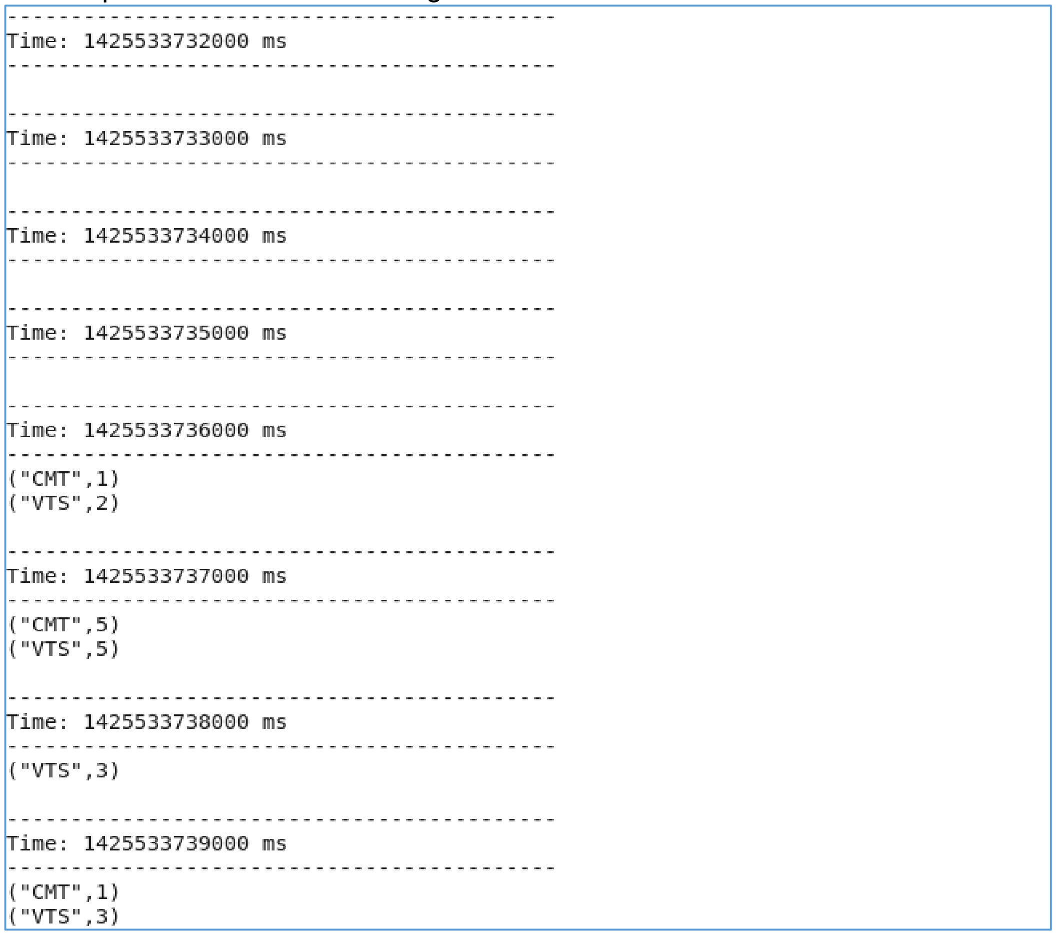
```
scala> import org.apache.spark._
import org.apache.spark._

scala> import org.apache.spark.streaming._
import org.apache.spark.streaming._

scala> import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.streaming.StreamingContext._

scala> val ssc = new StreamingContext(sc,Seconds(1))
ssc: org.apache.spark.streaming.StreamingContext = org.apache.spark.streaming.St
reamingContext@8299d690

scala> val lines = ssc.socketTextStream("localhost",7777)
lines: org.apache.spark.streaming.dstream.ReceiverInputDStream[String] = org.apa
che.spark.streaming.dstream.SocketInputDStream@65258611

scala> val pass = lines.map(_.split(",")).
     |            map(pass=>(pass(15),pass(7).toInt)).
     |            reduceByKey(_+_)
pass: org.apache.spark.streaming.dstream.DStream[(String, Int)] = org.apache.spa
rk.streaming.dstream.ShuffledDStream@a5833b

scala> pass.print()

scala> ssc.start()

scala> ssc.awaitTermination()
```

The first 3 lines import the required packages. Then a Spark streaming context ('ssc') is instantiated using the Spark Context ('sc') and using a 1 second window. Therefore, the data stream is divided into 1 second batches and each batch becomes an RDD. This was intentionally set to a high value to make it easier to view the data during execution. The program then connects to localhost socket 7777 which matches with the settings in the previous standalone Python program. The RDD created is 'lines'.

Processing on 'lines' starts with comma delimited parsing followed by a mapping of 2 values - 'pass(15)' which is the 'vendor' and 'pass(7)' which is the passenger count. This is then reduced by key resulting in a summary of number of passengers by vendor. This is printed out ('pass.print()') but could also be written to a file.

Finally, the process is started using the last two commands.

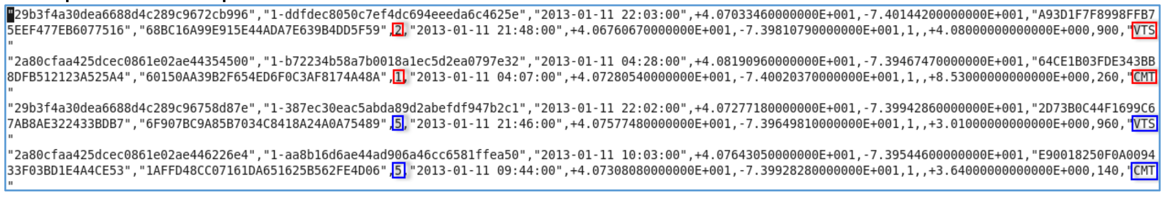
f) The output looks like the following.



```
-------------------------------------------
Time: 1425533732000 ms
-------------------------------------------

-------------------------------------------
Time: 1425533733000 ms
-------------------------------------------

-------------------------------------------
Time: 1425533734000 ms
-------------------------------------------

-------------------------------------------
Time: 1425533735000 ms
-------------------------------------------

-------------------------------------------
Time: 1425533736000 ms
-------------------------------------------
("CMT",1)
("VTS",2)

-------------------------------------------
Time: 1425533737000 ms
-------------------------------------------
("CMT",5)
("VTS",5)

-------------------------------------------
Time: 1425533738000 ms
-------------------------------------------
("VTS",3)

-------------------------------------------
Time: 1425533739000 ms
-------------------------------------------
("CMT",1)
("VTS",3)
```

It will take a few cycles for the connection to be recognized, then the data will be sent. In this case, 2 rows per second of taxi trip data is received in a 1 second window.

g) Compare the output it to the raw data.



```
"29b3f4a30dea6688d4c289c9672cb996","1-ddfdec8050c7ef4dc694eeeda6c4625e","2013-01-11 22:03:00",+4.07033460000000E+001,-7.40144200000000E+001,"A93D1F7F8998FFB7
5EEF477EB6077516","68BC16A99E915E44ADA7E639B4DD5F59",2,"2013-01-11 21:48:00",+4.06760670000000E+001,-7.39810790000000E+001,1,,+4.08000000000000E+000,900,"VTS
"
"2a80cfaa425dcec0861e02ae44354500","1-b72234b58a7b0018a1ec5d2ea0797e32","2013-01-11 04:28:00",+4.08190960000000E+001,-7.39467470000000E+001,"64CE1B03FDE343BB
8DFB512123A525A4","60150AA39B2F654ED6F0C3AF8174A48A",1,"2013-01-11 04:07:00",+4.07280540000000E+001,-7.40020370000000E+001,1,,+8.53000000000000E+000,260,"CMT
"
"29b3f4a30dea6688d4c289c96758d87e","1-387ec30eac5abda89d2abefdf947b2c1","2013-01-11 22:02:00",+4.07277180000000E+001,-7.39942860000000E+001,"2D73B0C44F1699C6
7AB8AE322433BDB7","6F907BC9A85B7034C8418A24A0A75489",5,"2013-01-11 21:46:00",+4.07577480000000E+001,-7.39649810000000E+001,1,,+3.01000000000000E+000,960,"VTS
"
"2a80cfaa425dcec0861e02ae446226e4","1-aa8b16d6ae44ad906a46cc6581ffea50","2013-01-11 10:03:00",+4.07643050000000E+001,-7.39544600000000E+001,"E90018250F0A0094
33F03BD1E4A4CE53","1AFFD48CC07161DA651625B562FE4D06",5,"2013-01-11 09:44:00",+4.07308080000000E+001,-7.39928280000000E+001,1,,+3.64000000000000E+000,140,"CMT
"
```

The red represents the data in the first 1 second window. Here we see that vendor 'CMT' had 1 passenger and vendor 'VTS' had 2 passengers. This matches with the output for the 1st batch. The blue represents the data in the second 1 second window. Here we see that vendor 'CMT' had 5 passengers and vendor 'VTS' has 5 passengers. This also matches with the output for the 2nd batch.

There is also the possibility where the first 1 second batch only captures one line. In this case, the beginning of the output could resemble this:

```
-------------------------------------------
Time: 1429580023000 ms
-------------------------------------------
("VTS",2)

-------------------------------------------
Time: 1429580024000 ms
-------------------------------------------
("CMT",1)
("VTS",5)
```

h) Please 'Control-C' out of each terminal window to stop and close the programs.

i) This could also be written as an application. Please refer to the previous labs for guidance.

j) Congratulations, you have complete this lab!

Great work and congratulations, you have completed this lab.

Screen captures in this document may vary slightly from yours.

# Lab 4 Getting Started with Spark SQL

In this lab, Spark SQL will be investigated. Spark SQL allows relational queries expressed in SQL, HiveQL or Scala to be executed using Spark. At the core of this component is a new type of RDD, SchemaRDD. SchemaRDDs are composed of Row objects, along with a schema that describes the data types of each column in the row. This approach will be used to achieve the same results in Lab 1 which used Scala. The same data for taxi trips will be used and the top 10 taxi medallion numbers based on the number of trips will be determined. This will be implemented in the Spark shell and a Spark application could also be created using this code. Please refer to the previous labs for guidance.

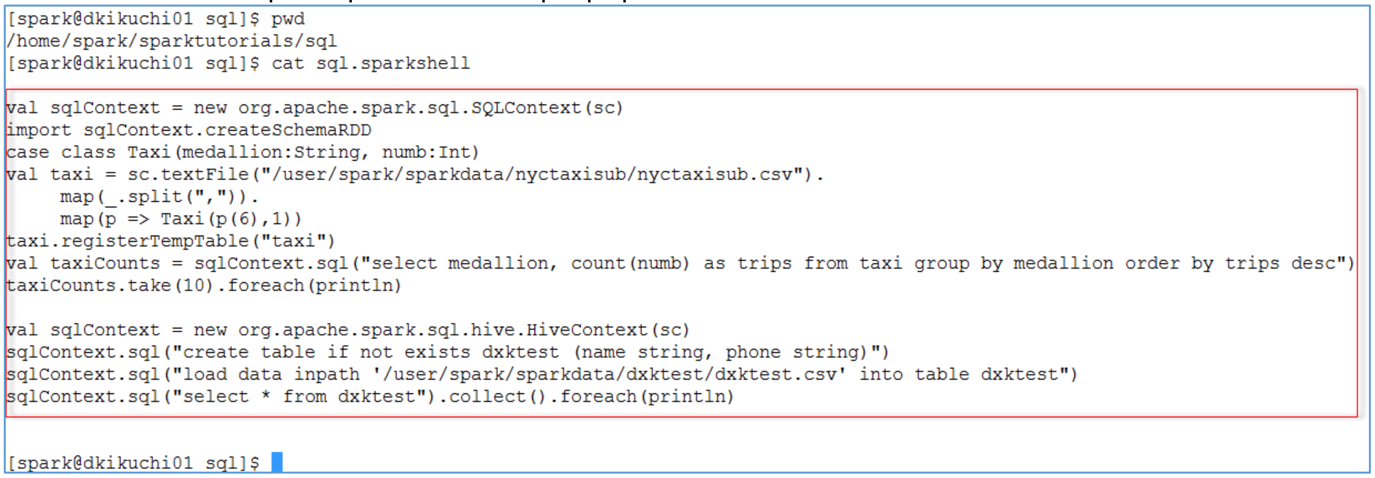
## Part 1 Initial Set Up with Sample Data and Code

If not already completed, please go through the steps in Lab 1, Part 1.

## Part 2 Using the Spark Shell

In this section, the Spark shell will be used to acquire the desired results, i.e. determination of the top 10 medallion numbers based on number of trips. The sample data contains a subset of taxi trips with hack license, medallion, pickup date/time, drop off date/time, pickup latitude/longitude, drop off latitude/longitude, passenger count, trip distance, trip time and other information.

The bold commands from this document can be copied and pasted to the Spark shell, and they are also available in '/home/spark/sparktutorials/sql/sql.sparkshell'.



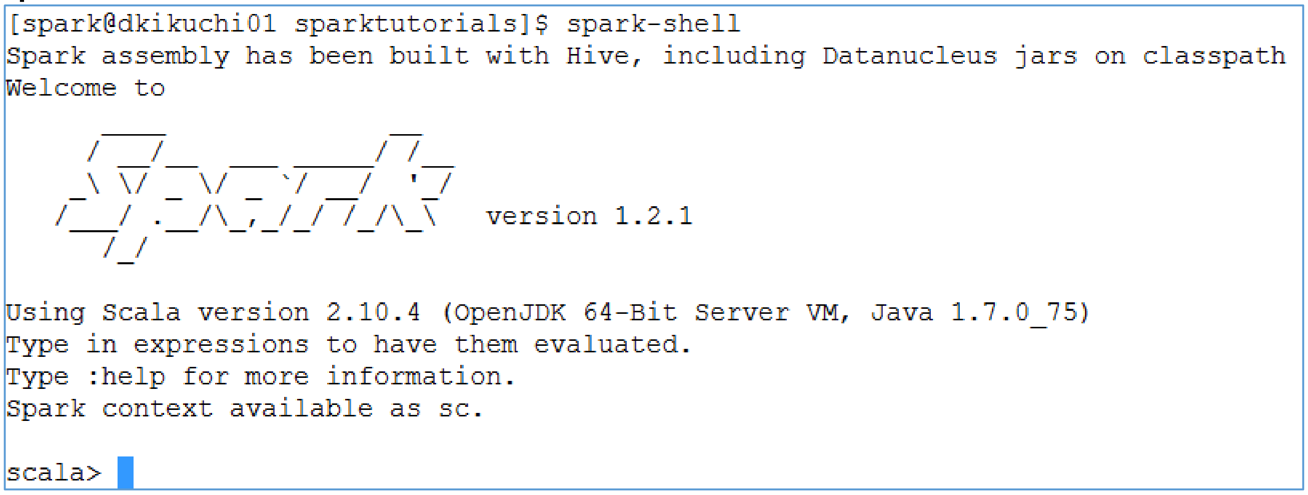
```
[spark@dkikuchi01 sql]$ pwd
/home/spark/sparktutorials/sql
[spark@dkikuchi01 sql]$ cat sql.sparkshell

val sqlContext = new org.apache.spark.sql.SQLContext(sc)
import sqlContext.createSchemaRDD
case class Taxi(medallion:String, numb:Int)
val taxi = sc.textFile("/user/spark/sparkdata/nyctaxisub/nyctaxisub.csv").
     map(_.split(",")).
     map(p => Taxi(p(6),1))
taxi.registerTempTable("taxi")
val taxiCounts = sqlContext.sql("select medallion, count(numb) as trips from taxi group by medallion order by trips desc")
taxiCounts.take(10).foreach(println)

val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)
sqlContext.sql("create table if not exists dxktest (name string, phone string)")
sqlContext.sql("load data inpath '/user/spark/sparkdata/dxktest/dxktest.csv' into table dxktest")
sqlContext.sql("select * from dxktest").collect().foreach(println)


[spark@dkikuchi01 sql]$
```

The inner red box show the contents of this file.

The Spark shell can be verbose. To view ERRORs only, a 'log4j.properties' file has been provided in 'home/spark/sparktutorials'. Please launch the 'spark-shell' in this subdirectory to suppress the verbose output.

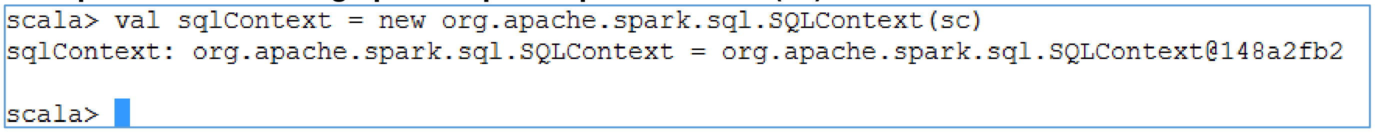
a) Please ensure that the BigInsights Version 4 environment has been started. Login as userid 'spark' or alternatively login as 'root' and then 'su - spark'.

b) Start a 'spark-shell' from '/home/spark/sparktutorials'.
**spark-shell**



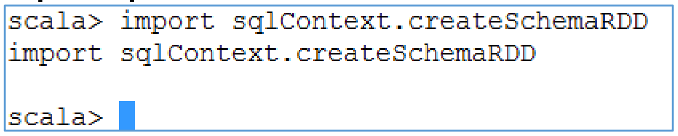
```
[spark@dkikuchi01 sparktutorials]$ spark-shell
Spark assembly has been built with Hive, including Datanucleus jars on classpath
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.2.1
      /_/

Using Scala version 2.10.4 (OpenJDK 64-Bit Server VM, Java 1.7.0_75)
Type in expressions to have them evaluated.
Type :help for more information.
Spark context available as sc.

scala>
```

c) Create a SQLContext from the SparkContext, sc.
**val sqlContext = new org.apache.spark.sql.SQLContext(sc)**



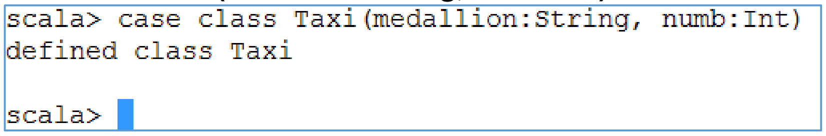
```
scala> val sqlContext = new org.apache.spark.sql.SQLContext(sc)
sqlContext: org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@148a2fb2

scala>
```

d) Import 'createSchemaRDD' which is used to implicitly convert an RDD to a SchemaRDD.
**import sqlContext.createSchemaRDD**



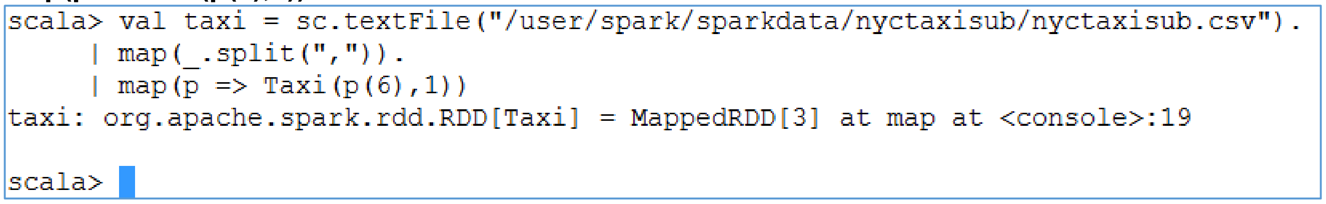
```
scala> import sqlContext.createSchemaRDD
import sqlContext.createSchemaRDD

scala>
```

e) Define the schema using a case class.
**case class Taxi(medallion:String, numb:Int)**



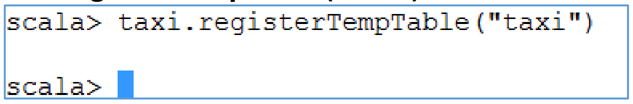
```
scala> case class Taxi(medallion:String, numb:Int)
defined class Taxi

scala>
```

f) Create an RDD of Taxi objects from and HDFS file.
**val taxi = sc.textFile("/user/spark/sparkdata/nyctaxisub/nyctaxisub.csv").**
**map(_.split(",")).**
**map(p => Taxi(p(6),1))**



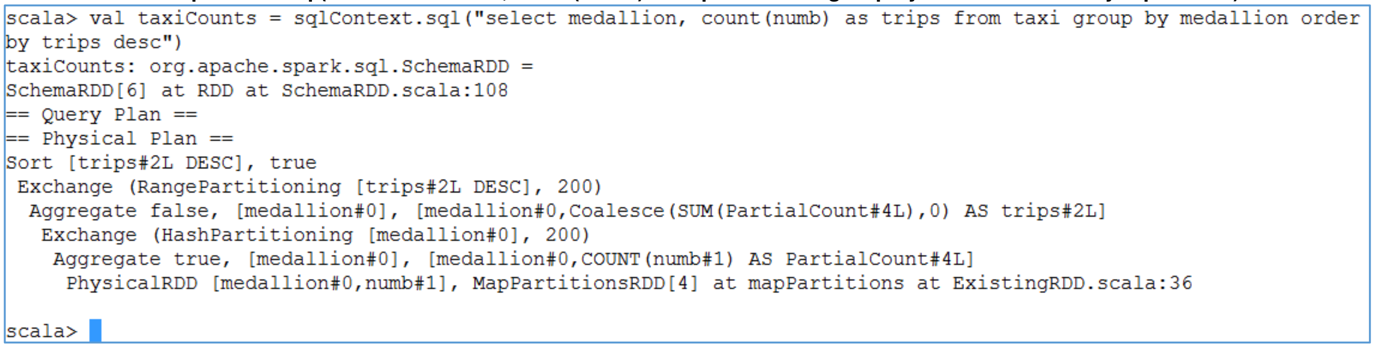
```
scala> val taxi = sc.textFile("/user/spark/sparkdata/nyctaxisub/nyctaxisub.csv").
     | map(_.split(",")).
     | map(p => Taxi(p(6),1))
taxi: org.apache.spark.rdd.RDD[Taxi] = MappedRDD[3] at map at <console>:19

scala>
```

g) Register ‘taxi’ as a table.
**taxi.registerTempTable("taxi")**



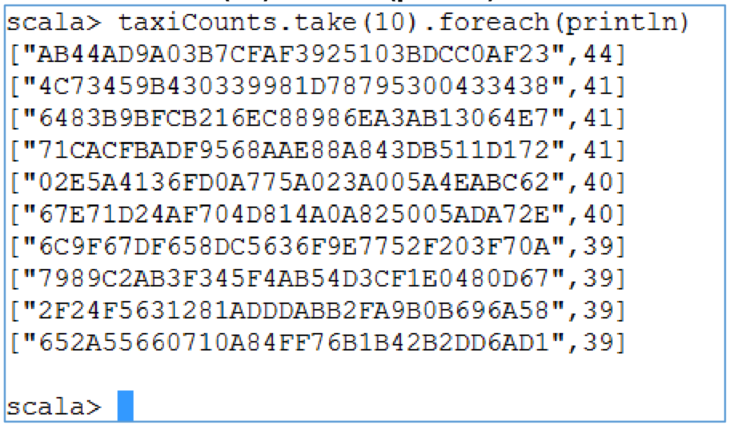
```
scala> taxi.registerTempTable("taxi")

scala>
```

h) Issue the SQL statement using the ‘sql’ method on ‘sqlContext’.
**val taxiCounts = sqlContext.sql("select medallion, count(numb) as trips from taxi group by medallion order by trips desc")**



```
scala> val taxiCounts = sqlContext.sql("select medallion, count(numb) as trips from taxi group by medallion order
by trips desc")
taxiCounts: org.apache.spark.sql.SchemaRDD =
SchemaRDD[6] at RDD at SchemaRDD.scala:108
== Query Plan ==
== Physical Plan ==
Sort [trips#2L DESC], true
 Exchange (RangePartitioning [trips#2L DESC], 200)
  Aggregate false, [medallion#0], [medallion#0,Coalesce(SUM(PartialCount#4L),0) AS trips#2L]
   Exchange (HashPartitioning [medallion#0], 200)
    Aggregate true, [medallion#0], [medallion#0,COUNT(numb#1) AS PartialCount#4L]
     PhysicalRDD [medallion#0,numb#1], MapPartitionsRDD[4] at mapPartitions at ExistingRDD.scala:36

scala>
```

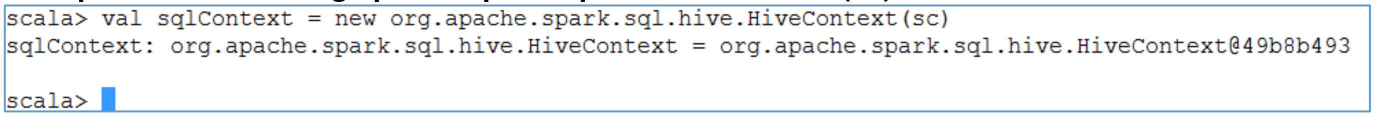
i) Print out the top 10 medallions with the most trips.
**taxiCounts.take(10).foreach(println)**



```
scala> taxiCounts.take(10).foreach(println)
["AB44AD9A03B7CFAF3925103BDCC0AF23",44]
["4C73459B430339981D78795300433438",41]
["6483B9BFCB216EC88986EA3AB13064E7",41]
["71CACFBADF9568AAE88A843DB511D172",41]
["02E5A4136FD0A775A023A005A4EABC62",40]
["67E71D24AF704D814A0A825005ADA72E",40]
["6C9F67DF658DC5636F9E7752F203F70A",39]
["7989C2AB3F345F4AB54D3CF1E0480D67",39]
["2F24F5631281ADDDABB2FA9B0B696A58",39]
["652A55660710A84FF76B1B42B2DD6AD1",39]

scala>
```

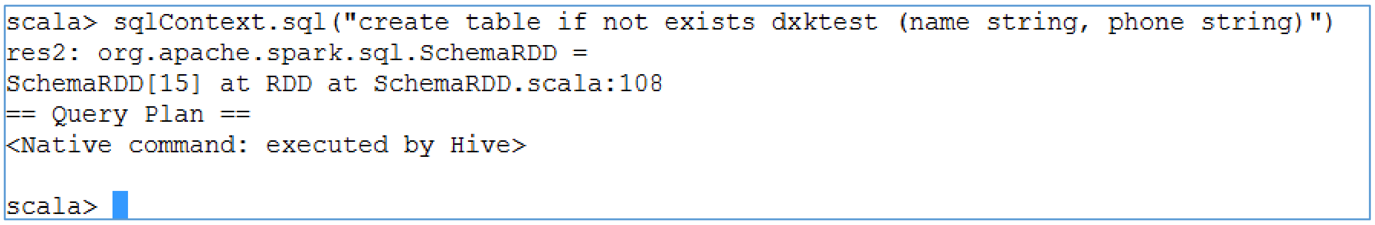
Note the results are the same as in Lab 1.

j) Spark SQL also suppors reading and writing data stored in Hive. In this case, a ‘HiveContext’ must be constructed from the ‘SparkContext’, sc. Issue the following in the Spark shell for Hive context.
**val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)**

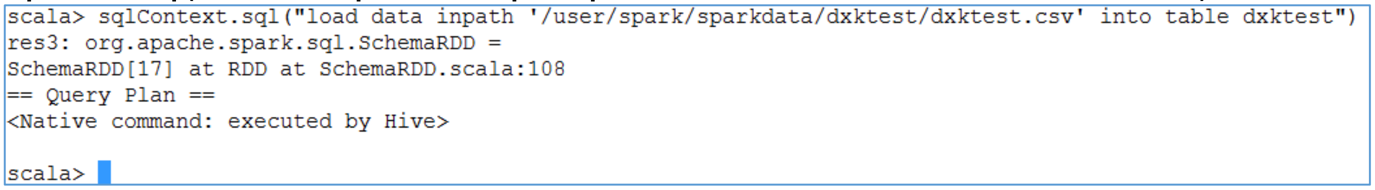


```
scala> val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)
sqlContext: org.apache.spark.sql.hive.HiveContext = org.apache.spark.sql.hive.HiveContext@49b8b493

scala>
```

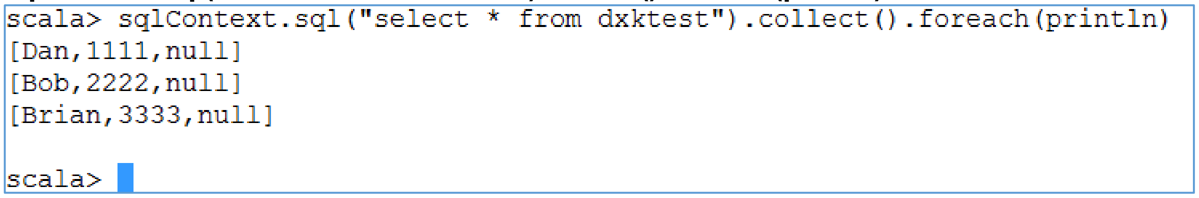
k) Create a Hive table ‘dxktest’.
**sqlContext.sql("create table if not exists dxktest (name string, phone string)")**


```
scala> sqlContext.sql("create table if not exists dxktest (name string, phone string)")
res2: org.apache.spark.sql.SchemaRDD =
SchemaRDD[15] at RDD at SchemaRDD.scala:108
== Query Plan ==
<Native command: executed by Hive>

scala>
```

l) Load data from HDFS into the newly create Hive table 'dxktest'.
**sqlContext.sql("load data inpath '/user/spark/sparkdata/dxktest/dxktest.csv' into table dxktest")**


```
scala> sqlContext.sql("load data inpath '/user/spark/sparkdata/dxktest/dxktest.csv' into table dxktest")
res3: org.apache.spark.sql.SchemaRDD =
SchemaRDD[17] at RDD at SchemaRDD.scala:108
== Query Plan ==
<Native command: executed by Hive>

scala>
```

m) Issue a SQL statement to query the contents.
**sqlContext.sql("select * from dxktest").collect().foreach(println)**


```
scala> sqlContext.sql("select * from dxktest").collect().foreach(println)
[Dan,1111,null]
[Bob,2222,null]
[Brian,3333,null]

scala>
```

Great work and congratulations, you have completed this lab.

Screen captures in this document may vary slightly from yours.

# Lab 5 Getting Started with IntelliJ & Spark

In this section, we will look at how to improve your productivity by using the popular IntelliJ IDEA IDE as a development tool. You will learn how to use IntelliJ to do all the development work on a windows workstation without accessing a BigInsights cluster at all. When your work is ready to test or deploy in a BigInsights cluster, we will show you the steps to build your project on Windows and submit the resulting JAR to run on a BigInsights cluster under yarn-cluster execution mode.

Many thanks to Steven Sit for creating this valuable asset. This is similar to Lab 1 however an the IntelliJ IDE will be used. The sample taxi data will also be needed.

## Part 1 Initial Set Up with Sample Data and Code

If not already completed, please go through the steps in Lab 1, Part 1.

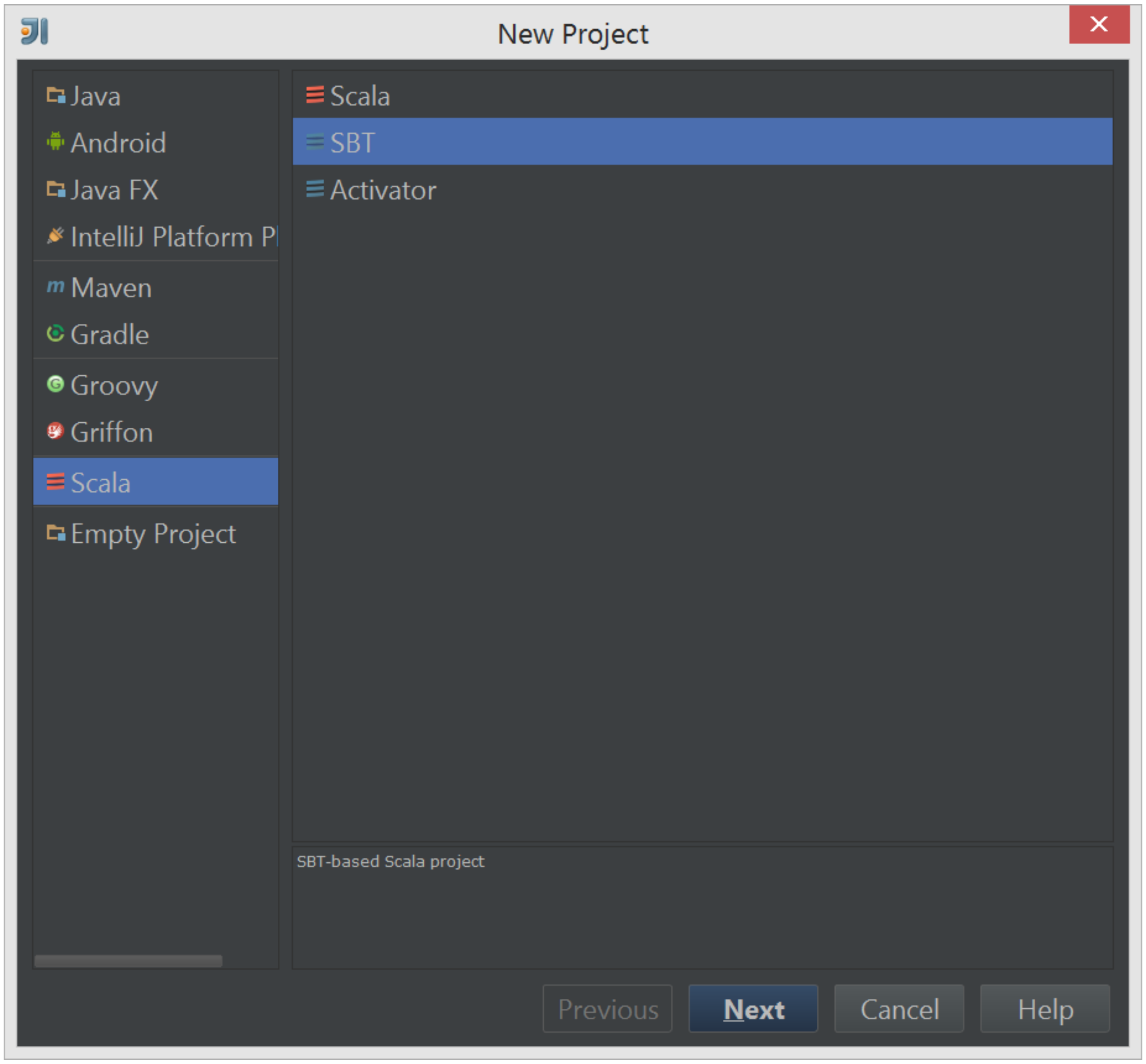
## Part 2 Installing and Preparing Your Workstation

1. Install IntelliJ Community Edition from jetbrains
   https://www.jetbrains.com/idea/download/
2. Install JDK 1.7 (not just JRE) if it is not yet installed
3. Install Scala and SBT plug-ins in IntelliJ follow the following instructions
   https://www.jetbrains.com/idea/help/installing-updating-and-uninstalling-repository-plugins.html
4. Download and install winutil.exe from Microsoft to work around a known bug in spark running on windows (IMPORTANT – this step is not needed if you are using a Mac)
   a. Download from here: http://public-repo-1.hortonworks.com/hdp-win-alpha/winutils.exe and put it under \yourpath\winutil\bin
   b. Use control panel ->system->advance system setting->environment variable to add HADOOP_HOME as a System Variable pointing to \yourpath\winutil (without the bin directory in the path)
5. After install plug-ins make sure the IDE is restarted to let it pick up the new settings

## Part 3 Create and configure a new project "sparktutorial2" using IntelliJ

Launch Intellij and create a new Scala SBT project using the main manual:

- New Project -> Scala->SBT

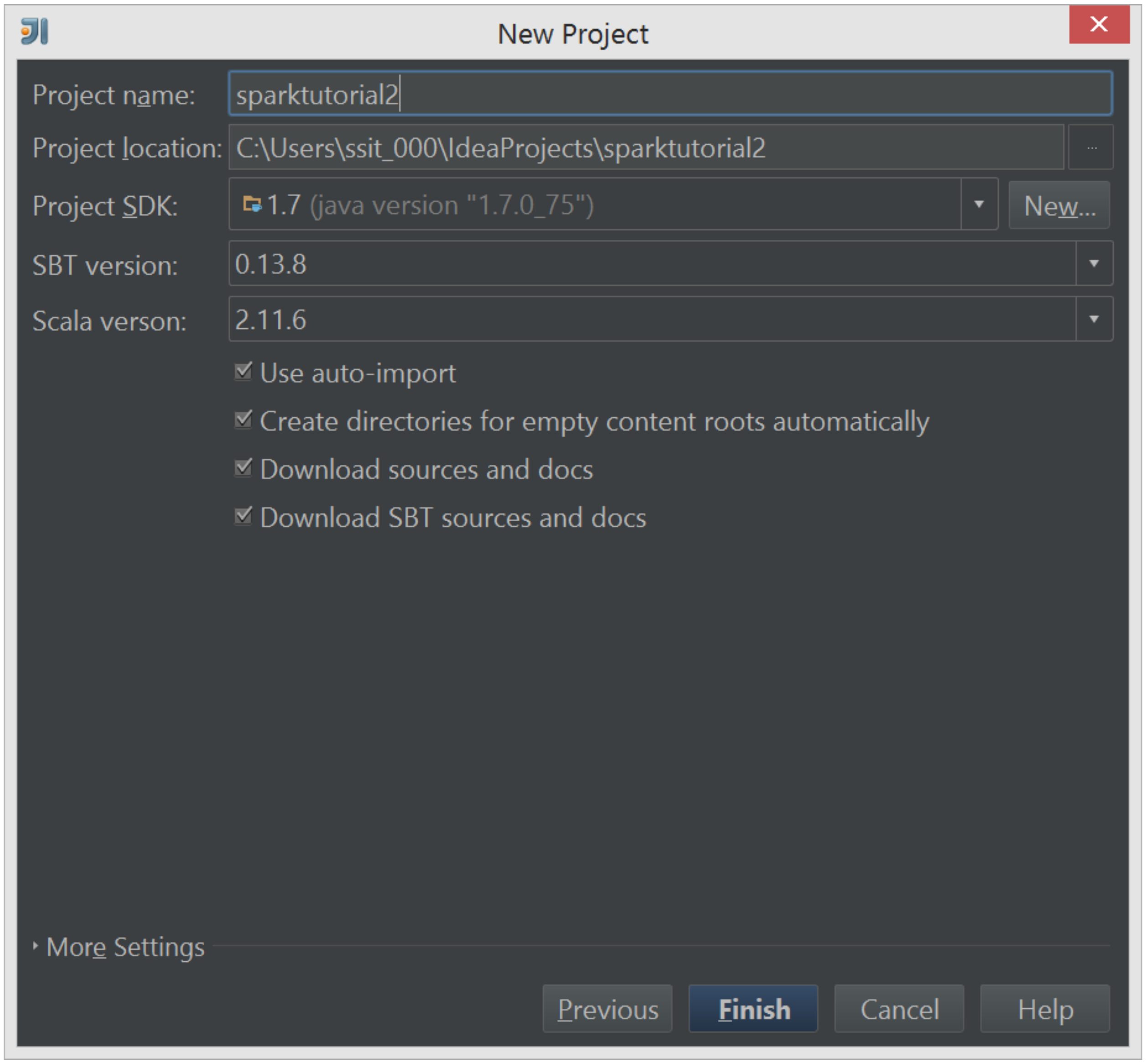


SBT-based Scala project

-

- **IMPORTANT** – make sure the "Use auto-import" option is checked (it is not checked by default), otherwise, the required Spark library will not get imported to the project. See screen below:

-


New Project

Project name: sparktutorial2

Project location: C:\Users\ssit_000\IdeaProjects\sparktutorial2

Project SDK: 1.7 (java version "1.7.0_75")

SBT version: 0.13.8

Scala verson: 2.11.6

☑ Use auto-import

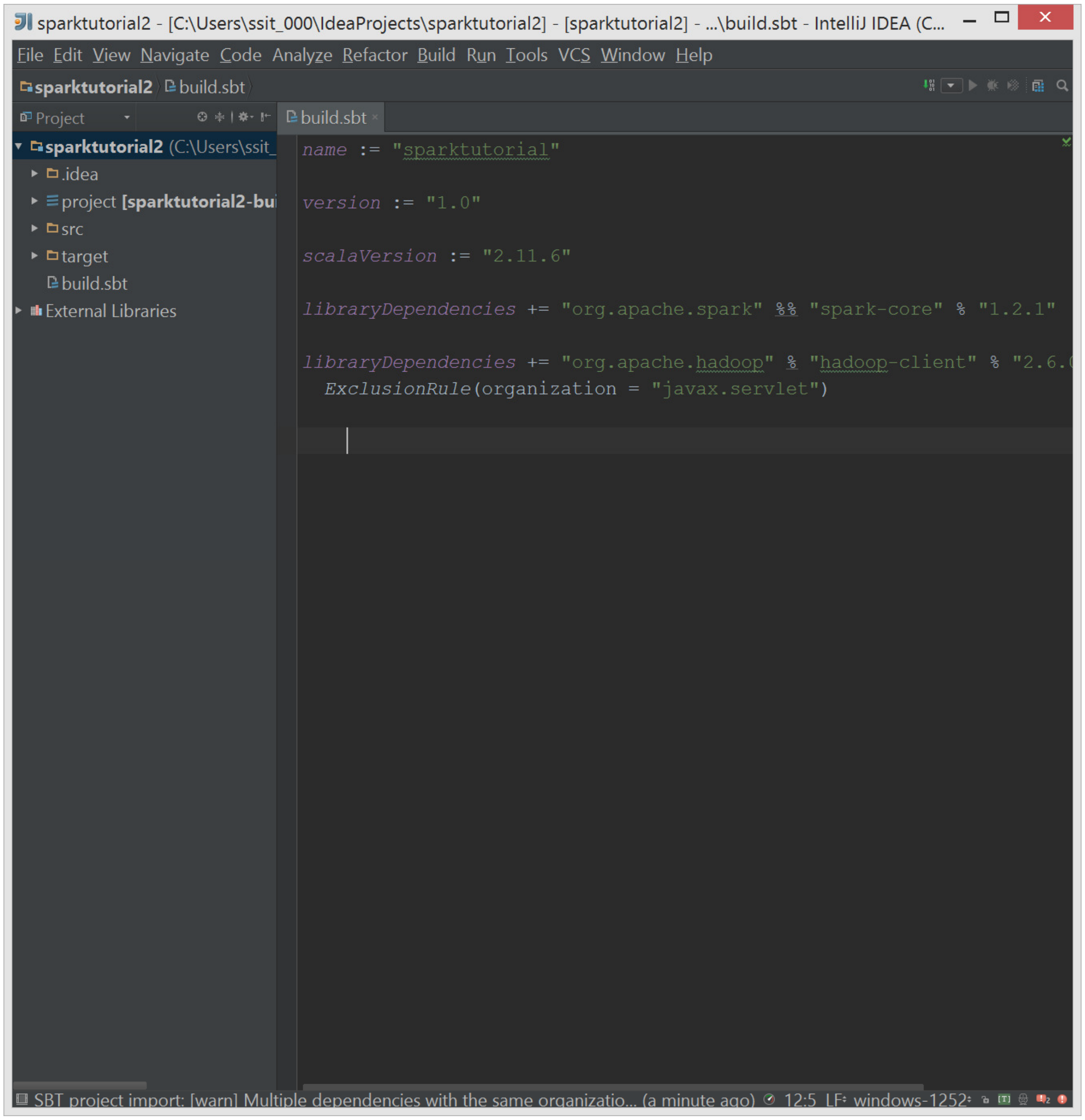
☑ Create directories for empty content roots automatically

☑ Download sources and docs

☑ Download SBT sources and docs

-

Now click-on the build.sbt file and add the spark and Hadoop dependencies based on the current BigInsights v4 levels of Hadoop and Spark support.



```
name := "sparktutorial"

version := "1.0"

scalaVersion := "2.11.6"

libraryDependencies += "org.apache.spark" %% "spark-core" % "1.2.1"

libraryDependencies += "org.apache.hadoop" % "hadoop-client" % "2.6.
  ExclusionRule(organization = "javax.servlet")
```

SBT project import: [warn] Multiple dependencies with the same organizatio... (a minute ago)  12:5  LF÷  windows-1252÷

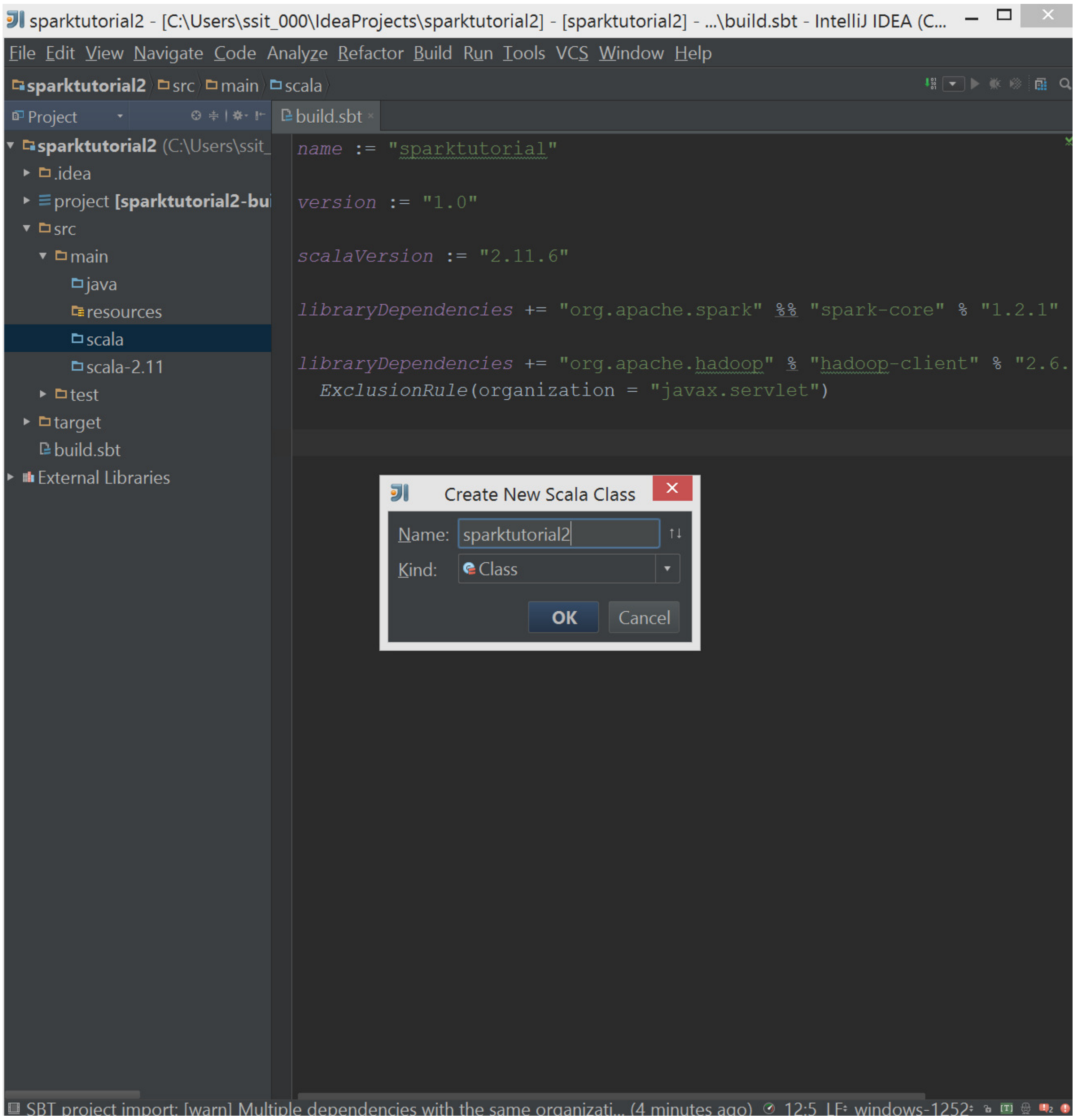
As you can see above, we need to add the line ExclusionRule to resolve a library conflict issue for the javax.servlet library which is in already included in the base libraries for Spark.

```
ExclusionRule(organization = "javax.servlet")
```

## Part 4 Create the new sparktutorial2.scala program

Now we are ready to create the sparktutorial2.scala source file which is a variation of the original sparktutorial.scala. It adds a few new things which should make your development efforts on windows much easier. You will be able to use IntelliJ for all the coding work, while able to build project using SBT tools and deploy your program to BigInisghts cluster as a yarn job.

First let's create a Scala Class file under src\main\scala directory in the project tree.


sparktutorial2 - [C:\Users\ssit_000\IdeaProjects\sparktutorial2] - [sparktutorial2] - ...\build.sbt - IntelliJ IDEA (C...
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
sparktutorial2 src main scala
Project
build.sbt
sparktutorial2 (C:\Users\ssit_
.idea
project [sparktutorial2-bui
src
main
java
resources
scala
scala-2.11
test
target
build.sbt
External Libraries
Create New Scala Class
OK
Cancel

```
name := "sparktutorial"

version := "1.0"

scalaVersion := "2.11.6"

libraryDependencies += "org.apache.spark" %% "spark-core" % "1.2.1"

libraryDependencies += "org.apache.hadoop" % "hadoop-client" % "2.6.
  ExclusionRule(organization = "javax.servlet")
```

Name: sparktutorial2
Kind: Class

SBT project import: [warn] Multiple dependencies with the same organizati... (4 minutes ago) 12:5 LF: windows-1252:

Once the file is created,  you can cut-and-paste the following code to the file.

```
/**
  * Created by ssit_000 on 4/18/2015.
  */

import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object sparktutorial2 {
  def main(args: Array[String]) {
    if (args.length < 2) {
      System.err.println("Usage: sparktutorial2 <host> <input_file>")
      System.exit(1)
    }

    val conf = new SparkConf()
      .setAppName("Spark Tutorial")
      .setMaster(args(0))
    val sc = new SparkContext(conf)
    val taxi = sc.textFile(args(1), 2).cache()
    val taxiMedCountsOneLine =
taxi.map(line=>line.split(',')).map(vals=>(vals(6),1)).reduceByKey(_+_
)
    for (pair<-taxiMedCountsOneLine.map(_.swap).top(10))
      println("Taxi Medallion %s had %s
Trips".format(pair._2,pair._1))
   }
}
```

As you can see, there are a few things added here.  First, we make the program to take two arguments as input.

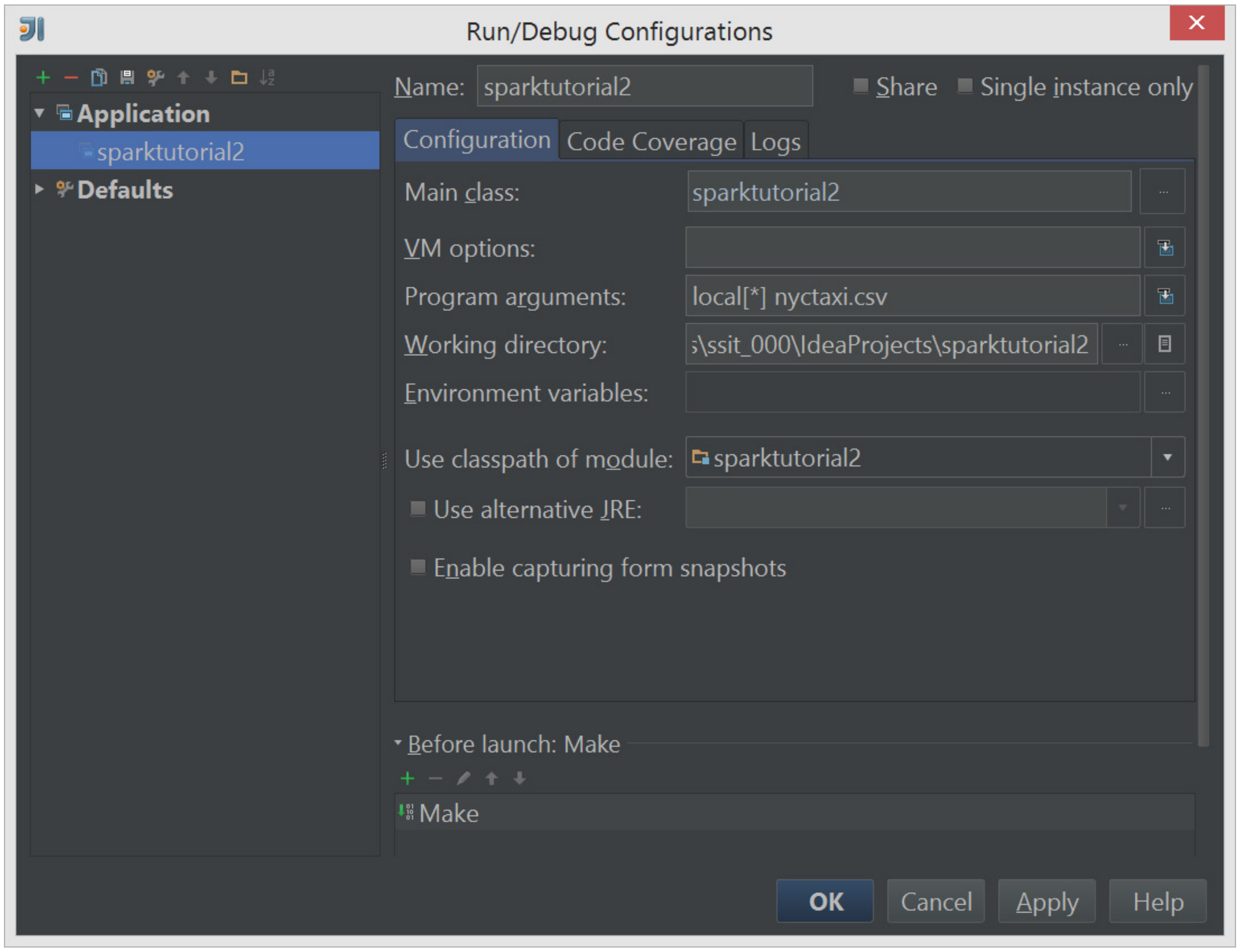
1. Execution mode.   This arguments will be used in the .setMaster statement for the master URL.  As you will find from the Spark documentation local[*] means running the program locally, and yarn-cluster means running under yarn in cluster mode.
2. Input file, which in this case should be nyctaxisub.csv

## Part 5 Configure and run the new sparktutorial2.scala program

Before running the program, let's copy the input file nyctaxisub.csv from the sparktutorials.zip to the project directory (e.g.  C:\Users\ssit_000\IdeaProjects\sparktutorial2)

Now try to run the program by right-mouse-click on sparktoturial2.scala and select Run

You will notice that the program should compile and run just fine, however it will complain about missing arguments.  To specific runtime arguments, got to Main Manual and select Run->Edit Configurations. Under the option “Program arguments”, specific **local[*] nyctaxisub.csv**.


Run/Debug Configurations

Application
sparktutorial2
Defaults

Name: sparktutorial2
Share
Single instance only

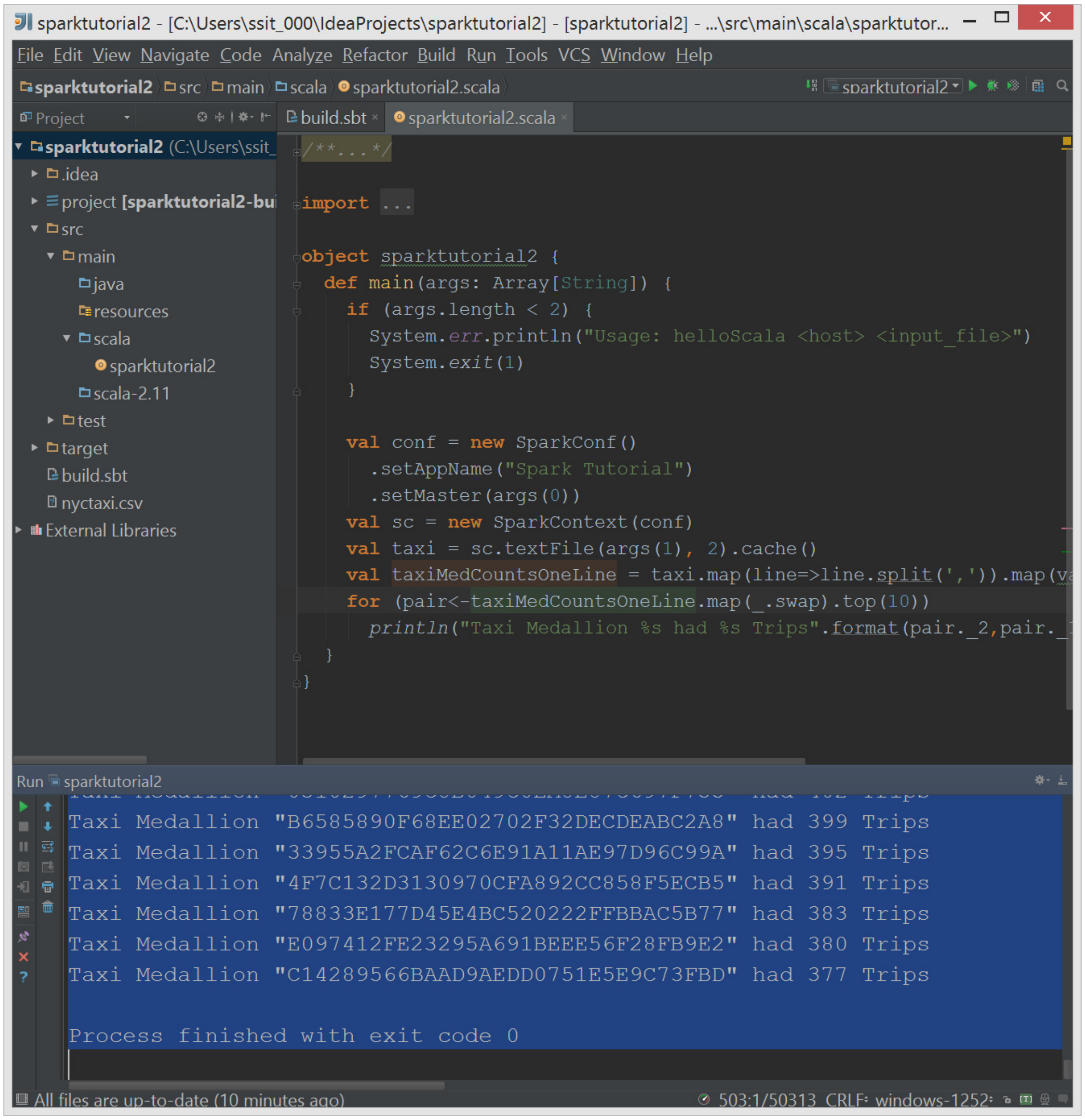
Configuration | Code Coverage | Logs

Main class: sparktutorial2
VM options:
Program arguments: local[*] nyctaxi.csv
Working directory: s\ssit_000\IdeaProjects\sparktutorial2
Environment variables:
Use classpath of module: sparktutorial2
Use alternative JRE:
Enable capturing form snapshots

Before launch: Make
Make

OK | Cancel | Apply | Help

Let’s run the program again.  This time it should complete with results show like the screen below.  On an i7 PC with SSD, it should take less than 1 minutes to execute.

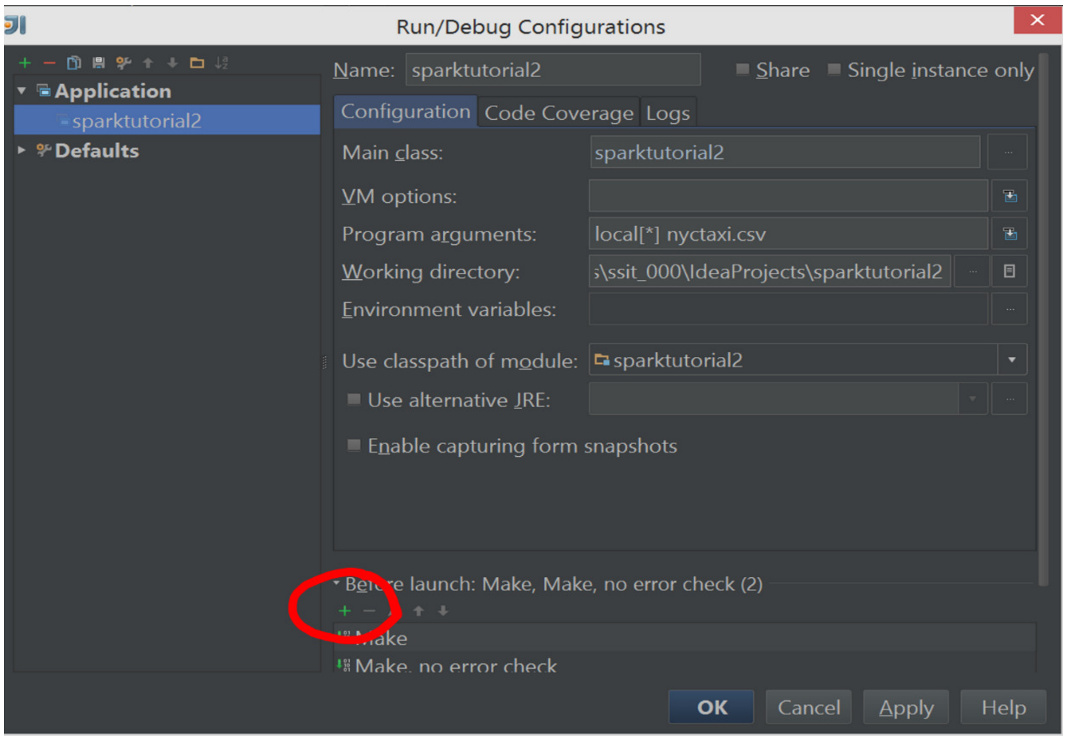

sparktutorial2 - [C:\Users\ssit_000\IdeaProjects\sparktutorial2] - [sparktutorial2] - ...\src\main\scala\sparktutor...
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
sparktutorial2 src main scala sparktutorial2.scala
Project
sparktutorial2 (C:\Users\ssit_
.idea
project [sparktutorial2-bui
src
main
java
resources
scala
sparktutorial2
scala-2.11
test
target
build.sbt
nyctaxi.csv
External Libraries
build.sbt
sparktutorial2.scala
Run sparktutorial2
All files are up-to-date (10 minutes ago)
503:1/50313 CRLF: windows-1252:

```
/**...*/

import ...

object sparktutorial2 {
  def main(args: Array[String]) {
    if (args.length < 2) {
      System.err.println("Usage: helloScala <host> <input_file>")
      System.exit(1)
    }

    val conf = new SparkConf()
      .setAppName("Spark Tutorial")
      .setMaster(args(0))
    val sc = new SparkContext(conf)
    val taxi = sc.textFile(args(1), 2).cache()
    val taxiMedCountsOneLine = taxi.map(line=>line.split(',')).map(v
    for (pair<-taxiMedCountsOneLine.map(_.swap).top(10))
      println("Taxi Medallion %s had %s Trips".format(pair._2,pair._
  }
}
```

```
Taxi Medallion "B6585890F68EE02702F32DECDEABC2A8" had 399 Trips
Taxi Medallion "33955A2FCAF62C6E91A11AE97D96C99A" had 395 Trips
Taxi Medallion "4F7C132D3130970CFA892CC858F5ECB5" had 391 Trips
Taxi Medallion "78833E177D45E4BC520222FFBBAC5B77" had 383 Trips
Taxi Medallion "E097412FE23295A691BEEE56F28FB9E2" had 380 Trips
Taxi Medallion "C14289566BAAD9AEDD0751E5E9C73FBD" had 377 Trips

Process finished with exit code 0
```

## Part 6 Build the JAR file for the program using SBT tools

Before using the SBT utility to build the JAR file, first we need to use IntelliJ to create all the necessary artifacts for our project. Go back to the Run-> Edit Configurations dialog and select the + like below to launch the SBT plugin to create the necessary files when you make and run the program


Run/Debug Configurations

Name: sparktutorial2

Main class: sparktutorial2

VM options:

Program arguments: local[*] nyctaxi.csv

Working directory: :\ssit_000\IdeaProjects\sparktutorial2

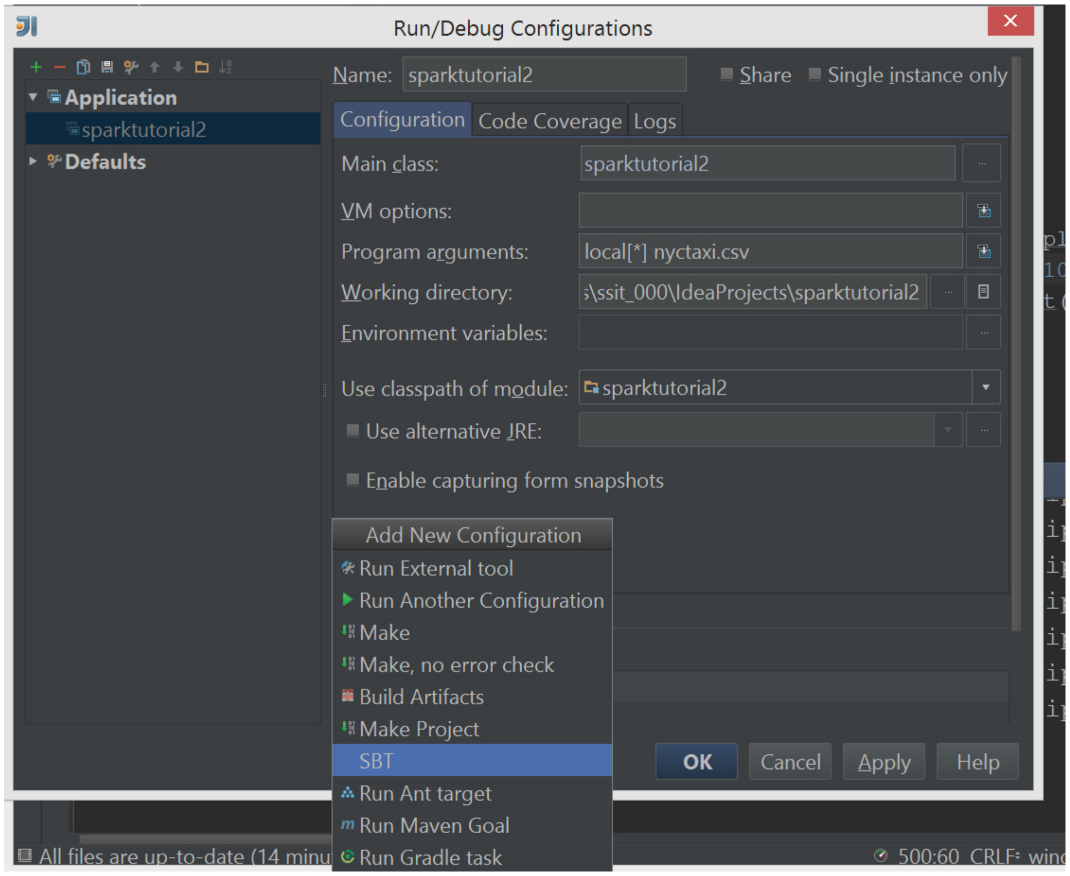
Environment variables:

Use classpath of module: sparktutorial2

Before launch: Make, Make, no error check (2)

Make

Make, no error check


Run/Debug Configurations

Name: sparktutorial2

Main class: sparktutorial2

VM options:

Program arguments: local[*] nyctaxi.csv

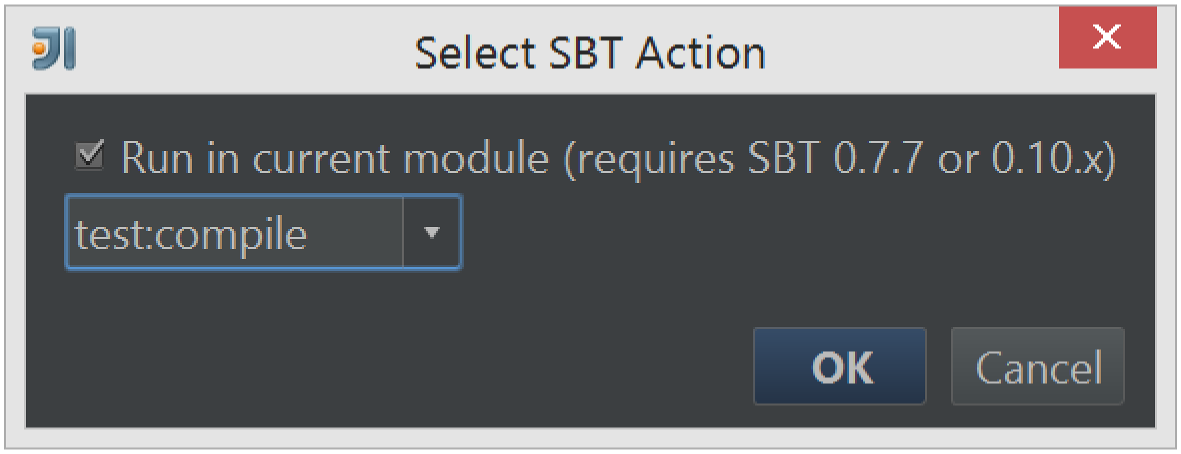
Working directory: :\ssit_000\IdeaProjects\sparktutorial2

Environment variables:

Use classpath of module: sparktutorial2

Add New Configuration
- Run External tool
- Run Another Configuration
- Make
- Make, no error check
- Build Artifacts
- Make Project
- SBT
- Run Ant target
- Run Maven Goal
- Run Gradle task

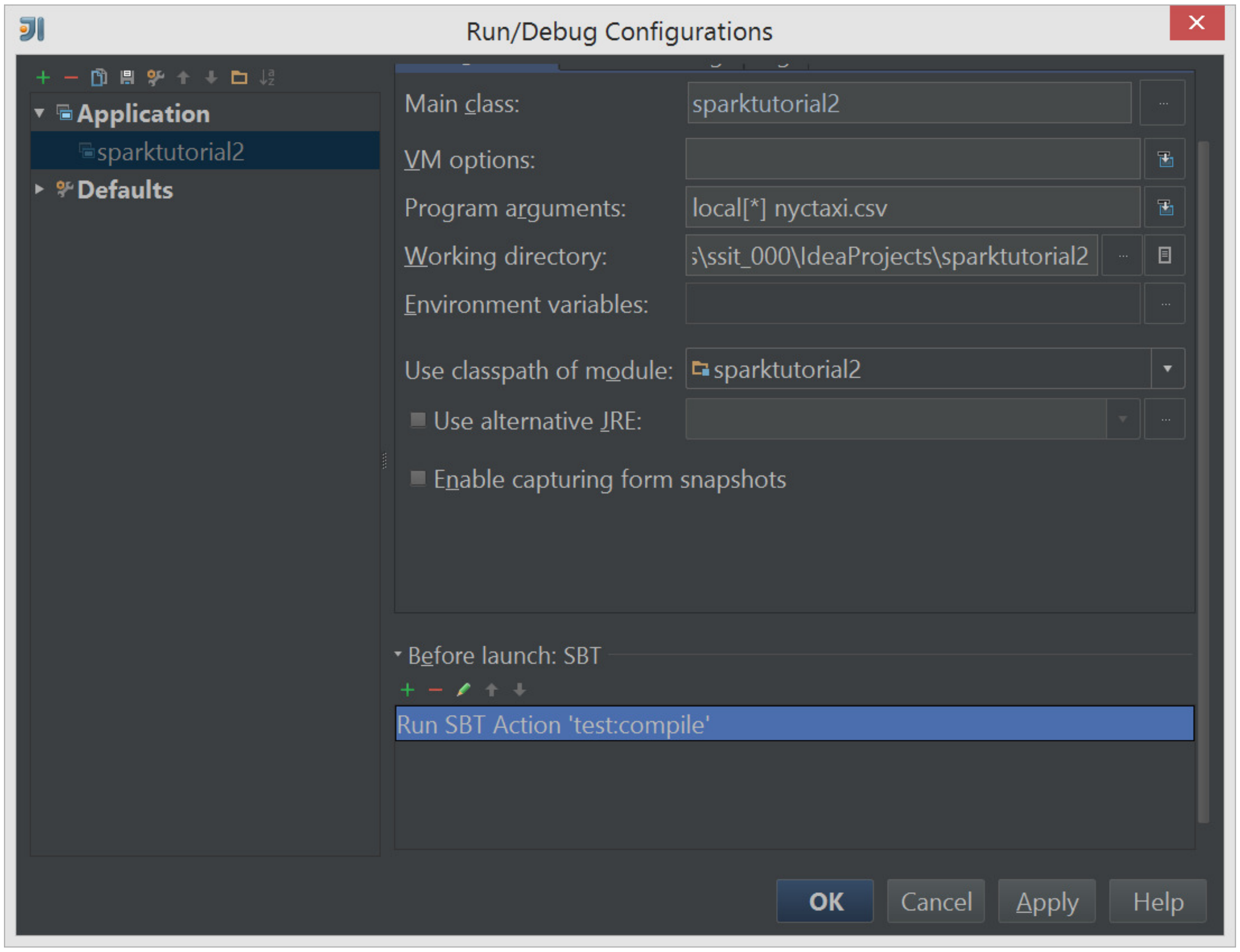
Make sure you select the compile option.



☑ Run in current module (requires SBT 0.7.7 or 0.10.x)

test:compile

You can also remove the “Make” action altogether. Your screen should look like below:



Main class: sparktutorial2

VM options:

Program arguments: local[*] nyctaxi.csv

Working directory: ;\ssit_000\IdeaProjects\sparktutorial2

Environment variables:

Use classpath of module: sparktutorial2

☐ Use alternative JRE:

☐ Enable capturing form snapshots

Before launch: SBT

Run SBT Action 'test:compile'

Now let's Run the program again. This time IntelliJ will also create the artifacts for the SBT built.

Then our next task is to use commands in a DOS windows and build the JAR file. Follow the following steps:

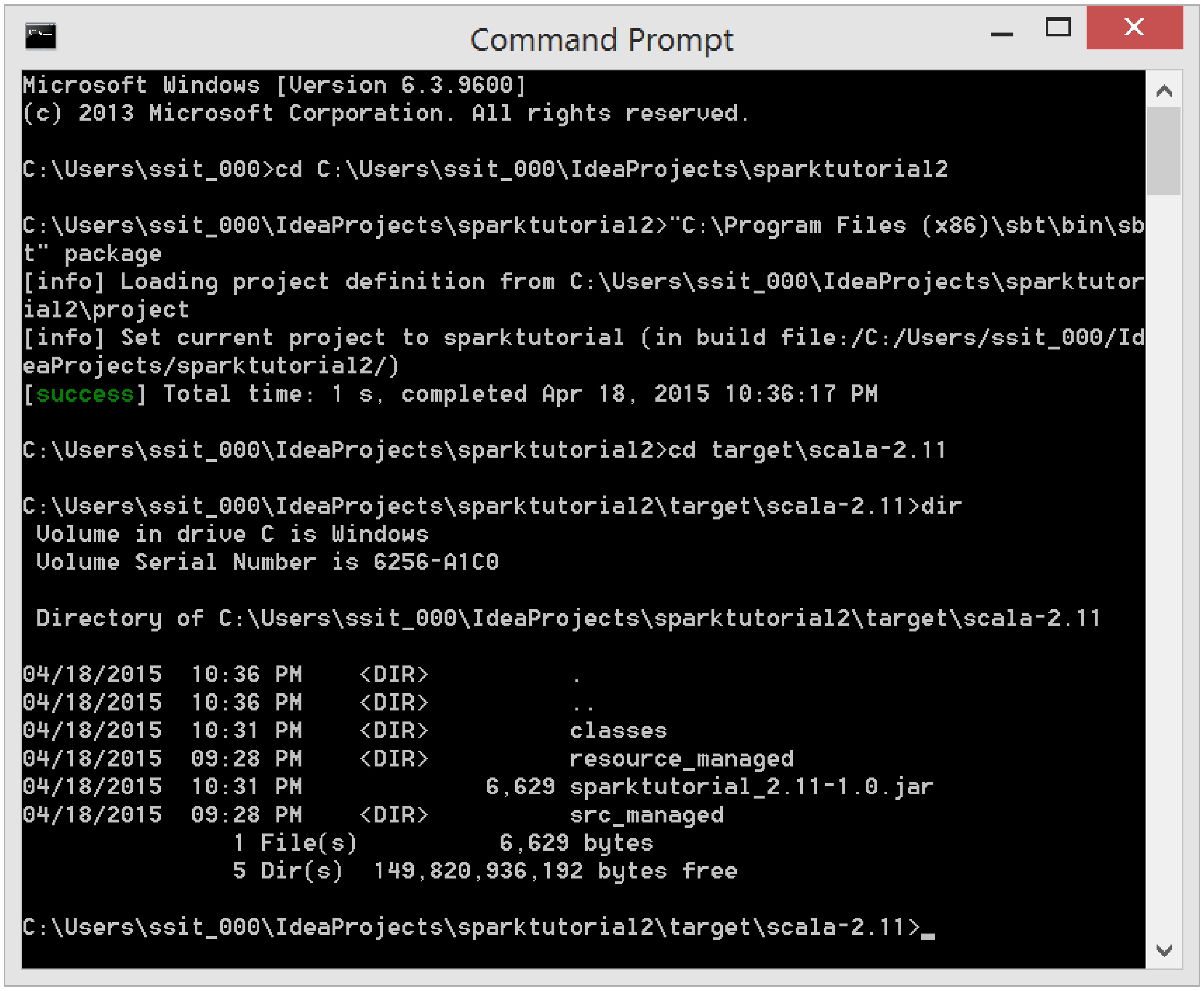
Change directory into the project directory. Which in my case is:

cd C:\Users\ssit_000\IdeaProjects\sparktutorial2

Build the JAR file using SBT tool under your SBT install directory. Which in my case is:

"C:\Program Files (x86)\sbt\bin\sbt" package

Now the JAR file is created under \target\scala-2.11. See screen below for these steps:



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\ssit_000>cd C:\Users\ssit_000\IdeaProjects\sparktutorial2

C:\Users\ssit_000\IdeaProjects\sparktutorial2>"C:\Program Files (x86)\sbt\bin\sb
t" package
[info] Loading project definition from C:\Users\ssit_000\IdeaProjects\sparktutor
ial2\project
[info] Set current project to sparktutorial (in build file:/C:/Users/ssit_000/Id
eaProjects/sparktutorial2/)
[success] Total time: 1 s, completed Apr 18, 2015 10:36:17 PM

C:\Users\ssit_000\IdeaProjects\sparktutorial2>cd target\scala-2.11

C:\Users\ssit_000\IdeaProjects\sparktutorial2\target\scala-2.11>dir
 Volume in drive C is Windows
 Volume Serial Number is 6256-A1C0

 Directory of C:\Users\ssit_000\IdeaProjects\sparktutorial2\target\scala-2.11

04/18/2015  10:36 PM    <DIR>          .
04/18/2015  10:36 PM    <DIR>          ..
04/18/2015  10:31 PM    <DIR>          classes
04/18/2015  09:28 PM    <DIR>          resource_managed
04/18/2015  10:31 PM             6,629 sparktutorial_2.11-1.0.jar
04/18/2015  09:28 PM    <DIR>          src_managed
               1 File(s)          6,629 bytes
               5 Dir(s)  149,820,936,192 bytes free

C:\Users\ssit_000\IdeaProjects\sparktutorial2\target\scala-2.11>_
```

You can upload the resulting JAR file to a Linux server which can be one of the nodes in the BigInsights cluster or an edge-node. You can use SFTP, for example, to upload the file so you can submit it as a spark job in yarn-cluster mode.

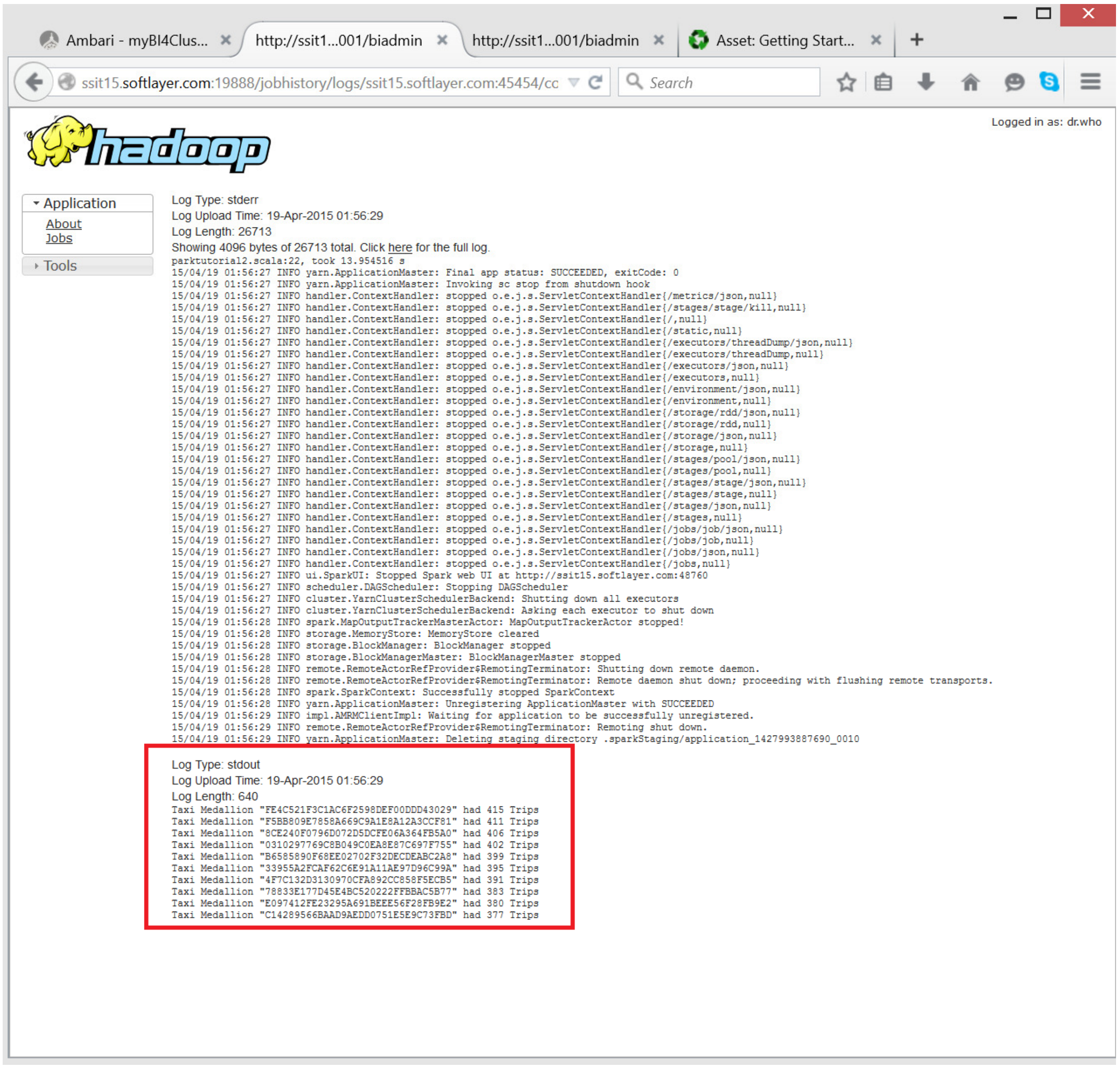
## Part 7 Run sparktutorial2 on BigInsights under yarn-cluster mode

In the example below, we used spark-submit to execute the program using yarn-cluster mode.  This time, the input file "nyctaxi.csv" by default should be stored in HDFS.  By default, program is looking for the path under the user's home directory in HDFS.

**$spark-submit --class "sparktutorial2" --master yarn-cluster sparktutorial_2.11-1.0.jar yarn-cluster nyctaxi.csv**

Finally, you can see the output of the program in the yarn AM UI.

Congratulations!  You are done with this part of the tutorial.  Happy programming in Spark/Scala using a "real" IDE.  ☺



Ambari - myBI4Clus...
http://ssit1...001/biadmin
http://ssit1...001/biadmin
Asset: Getting Start...
ssit15.softlayer.com:19888/jobhistory/logs/ssit15.softlayer.com:45454/cc
Search
Logged in as: dr.who
Application
About
Jobs
Tools

Log Type: stderr
Log Upload Time: 19-Apr-2015 01:56:29
Log Length: 26713
Showing 4096 bytes of 26713 total. Click here for the full log.

```
parktutorial2.scala:22, took 13.954516 s
15/04/19 01:56:27 INFO yarn.ApplicationMaster: Final app status: SUCCEEDED, exitCode: 0
15/04/19 01:56:27 INFO yarn.ApplicationMaster: Invoking sc stop from shutdown hook
15/04/19 01:56:27 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/metrics/json,null}
15/04/19 01:56:27 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages/stage/kill,null}
15/04/19 01:56:27 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/,null}
15/04/19 01:56:27 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/static,null}
15/04/19 01:56:27 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/executors/threadDump/json,null}
15/04/19 01:56:27 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/executors/threadDump,null}
15/04/19 01:56:27 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/executors/json,null}
15/04/19 01:56:27 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/executors,null}
15/04/19 01:56:27 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/environment/json,null}
15/04/19 01:56:27 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/environment,null}
15/04/19 01:56:27 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/storage/rdd/json,null}
15/04/19 01:56:27 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/storage/rdd,null}
15/04/19 01:56:27 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/storage/json,null}
15/04/19 01:56:27 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/storage,null}
15/04/19 01:56:27 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages/pool/json,null}
15/04/19 01:56:27 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages/pool,null}
15/04/19 01:56:27 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages/stage/json,null}
15/04/19 01:56:27 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages/stage,null}
15/04/19 01:56:27 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages/json,null}
15/04/19 01:56:27 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages,null}
15/04/19 01:56:27 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/jobs/job/json,null}
15/04/19 01:56:27 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/jobs/job,null}
15/04/19 01:56:27 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/jobs/json,null}
15/04/19 01:56:27 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/jobs,null}
15/04/19 01:56:27 INFO ui.SparkUI: Stopped Spark web UI at http://ssit15.softlayer.com:48760
15/04/19 01:56:27 INFO scheduler.DAGScheduler: Stopping DAGScheduler
15/04/19 01:56:27 INFO cluster.YarnClusterSchedulerBackend: Shutting down all executors
15/04/19 01:56:27 INFO cluster.YarnClusterSchedulerBackend: Asking each executor to shut down
15/04/19 01:56:28 INFO spark.MapOutputTrackerMasterActor: MapOutputTrackerActor stopped!
15/04/19 01:56:28 INFO storage.MemoryStore: MemoryStore cleared
15/04/19 01:56:28 INFO storage.BlockManager: BlockManager stopped
15/04/19 01:56:28 INFO storage.BlockManagerMaster: BlockManagerMaster stopped
15/04/19 01:56:28 INFO remote.RemoteActorRefProvider$RemotingTerminator: Shutting down remote daemon.
15/04/19 01:56:28 INFO remote.RemoteActorRefProvider$RemotingTerminator: Remote daemon shut down; proceeding with flushing remote transports.
15/04/19 01:56:28 INFO spark.SparkContext: Successfully stopped SparkContext
15/04/19 01:56:28 INFO yarn.ApplicationMaster: Unregistering ApplicationMaster with SUCCEEDED
15/04/19 01:56:29 INFO impl.AMRMClientImpl: Waiting for application to be successfully unregistered.
15/04/19 01:56:29 INFO remote.RemoteActorRefProvider$RemotingTerminator: Remoting shut down.
15/04/19 01:56:29 INFO yarn.ApplicationMaster: Deleting staging directory .sparkStaging/application_1427993887690_0010
```

Log Type: stdout
Log Upload Time: 19-Apr-2015 01:56:29
Log Length: 640

```
Taxi Medallion "FE4C521F3C1AC6F2598DEF00DDD43029" had 415 Trips
Taxi Medallion "F5BB809E7858A669C9A1E8A12A3CCF81" had 411 Trips
Taxi Medallion "8CE240F0796D072D5DCFE06A364FB5A0" had 406 Trips
Taxi Medallion "0310297769C8B049C0EA8E87C697F755" had 402 Trips
Taxi Medallion "B6585890F68EE02702F32DECDEABC2A8" had 399 Trips
Taxi Medallion "33955A2FCAF62C6E91A11AE97D96C99A" had 395 Trips
Taxi Medallion "4F7C132D3130970CFA892CC858F5ECB5" had 391 Trips
Taxi Medallion "78833E177D45E4BC520222FFBBAC5B77" had 383 Trips
Taxi Medallion "E097412FE23295A691BEEE56F28FB9E2" had 380 Trips
Taxi Medallion "C14289566BAAD9AEDD0751E5E9C73FBD" had 377 Trips
```

## NOTES