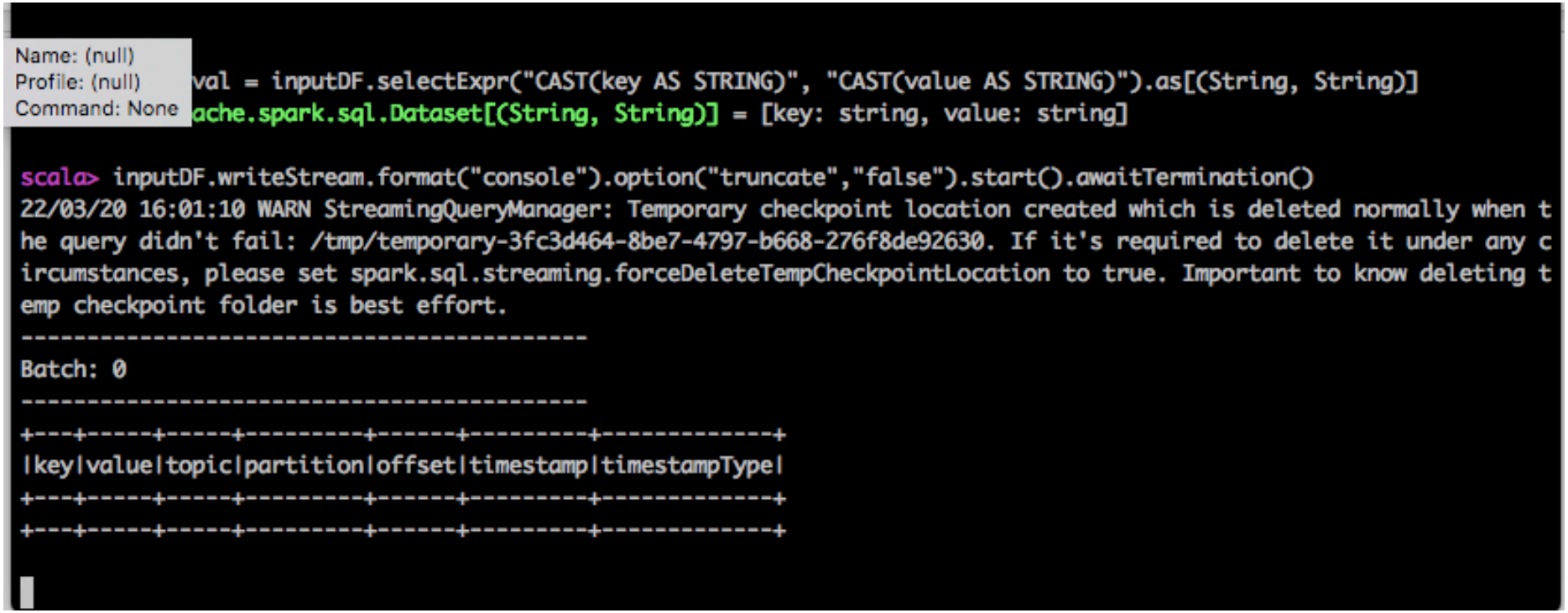


Open a terminal and send some message from the Producer Console.

#/opt/kafka/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic events

```
[root@303b68143644 /]# /opt/kafka/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic eve
nts
>Henry
>Learning Well
>Again
>Again
>
```

You can verify the result on the Spark Shell.

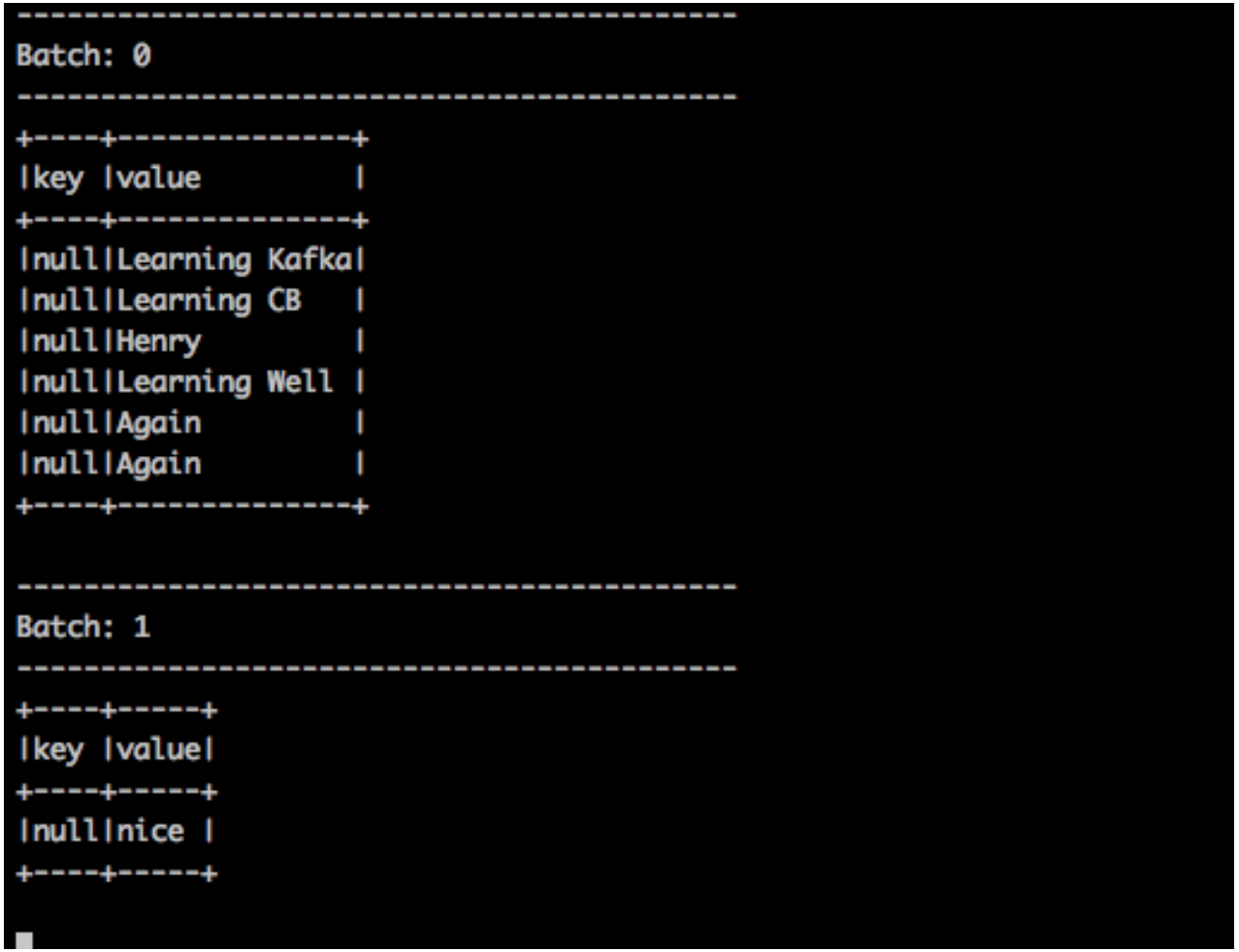
```
scala> -------------------------------------------
Batch: 4
-------------------------------------------
+----+----------------+------+---------+------+--------------------+-------------+
| key|           value| topic|partition|offset|           timestamp|timestampType|
+----+----------------+------+---------+------+--------------------+-------------+
|null|[41 67 61 69 6E]|events|        0|     5|2021-03-06 10:11:...|            0|
+----+----------------+------+---------+------+--------------------+-------------+
```

Spark sees the data in binary format, but we know it is actually text data. Let's first convert the messages to strings.

Quit the shell and execute again.

```
#import spark.implicits._

#val inputDF = spark.readStream.format("kafka").option("kafka.bootstrap.servers",
"localhost:9092").option("subscribe", "events").option("startingOffsets",
"earliest").option("checkpointLocation", "\tmp").load()

#val myval = inputDF.selectExpr("CAST(key AS STRING)", "CAST(value AS
STRING)").as[(String, String)]


#myval.writeStream.format("console").option("truncate","false").start().awaitTermination(
)
```

Type some messages on the producer console.



Batch: 0

+----+--------------+
|key |value |
+----+--------------+
null	Learning Kafka
null	Learning CB
null	Henry
null	Learning Well
null	Again
null	Again
+----+--------------+

Batch: 1

+----+-----+
|key |value|
+----+-----+
|null|nice |
+----+-----+

Now let us perform a word count on the text inserted in the topic.

Create another topic: (logs)

```
#/opt/kafka/bin/kafka-topics.sh --create \
  --bootstrap-server localhost:9092 \
  --partitions 8 \
  --topic logs
```

Import org.apache.spark.sql.functions._, it includes UDF's that i need to use import org.apache.spark.sql.functions._

```
#import spark.implicits._
#import org.apache.spark.sql.functions._
```

```
val inputDF = spark.readStream.format("kafka").option("kafka.bootstrap.servers", "localhost:9092").option("subscribe", "logs").option("startingOffsets", "earliest").option("checkpointLocation", "\tmp\\learning").load()

val myval = inputDF.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)").as[(String, String)]
```

Next split each of the line into words using split function. This will create a new DataFrame with words column, each words column would have array of words for that line

```
val wordsDF = myval.select(split(myval("value")," ").alias("words"))
```

Next use explode transformation to convert the words array into a dataframe with word column. This is equivalent of using flatMap() method on RDD

```
val wordDF = wordsDF.select(explode(wordsDF("words")).alias("word"))
```

Now you have data frame with each line containing single word in the file. So group the data frame based on word and count the occurrence of each word

```
val wordCountDF = wordDF.groupBy("word").count()
```

```
scala> val inputDF = spark.readStream.format("kafka").option("kafka.bootstrap.servers", "localhost:9092").option("su
bscribe", "logs").option("startingOffsets", "earliest").option("checkpointLocation", "\tmp\\learning").load()
inputDF: org.apache.spark.sql.DataFrame = [key: binary, value: binary ... 5 more fields]

scala> val myval = inputDF.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)").as[(String, String)]
myval: org.apache.spark.sql.Dataset[(String, String)] = [key: string, value: string]

scala> val wordsDF = myval.select(split(myval("value")," ").alias("words"))
wordsDF: org.apache.spark.sql.DataFrame = [words: array<string>]

scala> val wordDF = wordsDF.select(explode(wordsDF("words")).alias("word"))
wordDF: org.apache.spark.sql.DataFrame = [word: string]

scala> val wordCountDF = wordDF.groupBy("word").count()
wordCountDF: org.apache.spark.sql.DataFrame = [word: string, count: bigint]
```

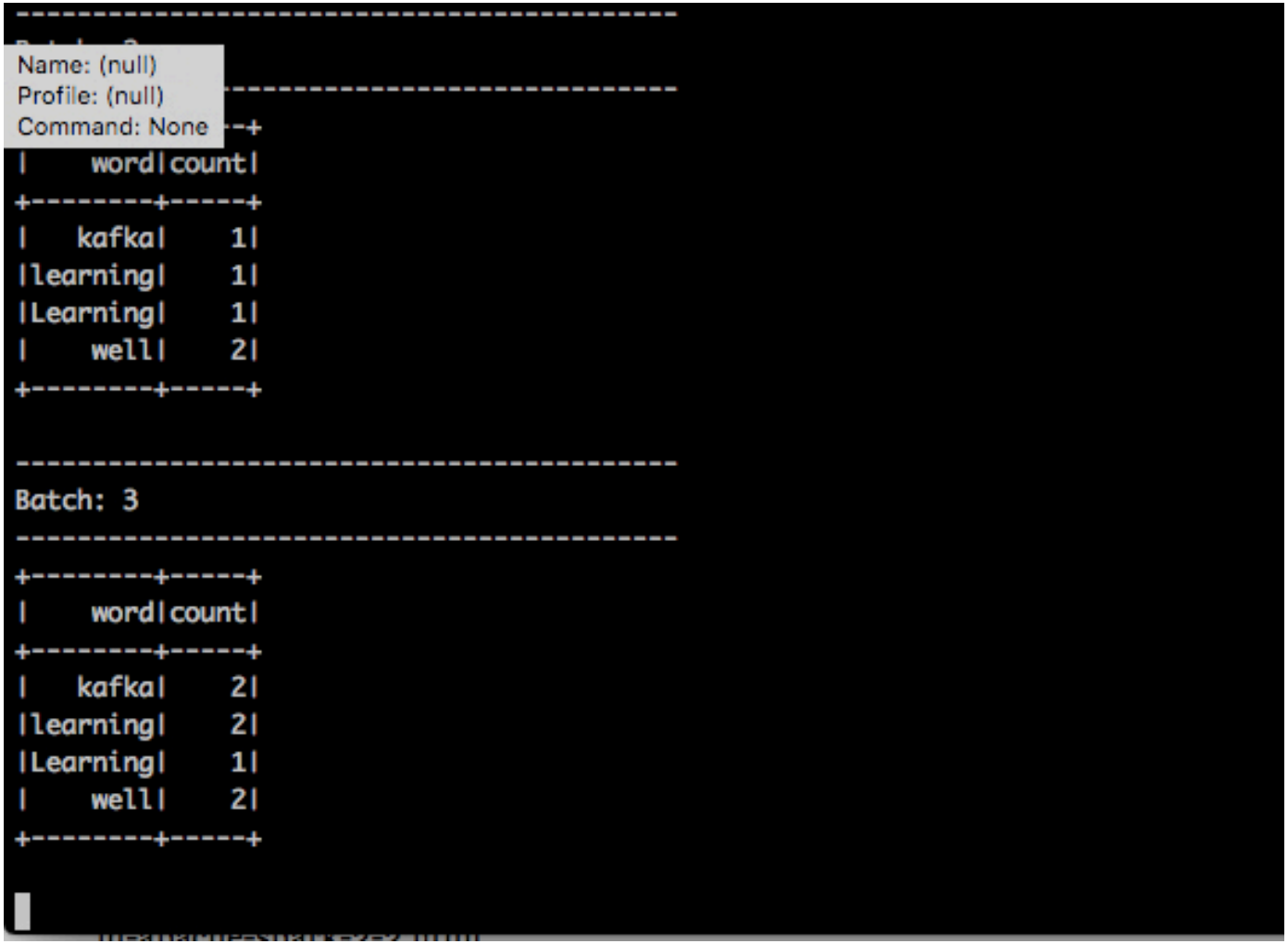
wordCountDF.writeStream.outputMode("complete").format("console").option("truncate","true").start().awaitTermination()

```
scala> wordCountDF.writeStream.outputMode("complete").format("console").option("truncate","true").start().awaitTermi
nation()
22/03/20 17:20:19 WARN StreamingQueryManager: Temporary checkpoint location created which is deleted normally when t
he query didn't fail: /tmp/temporary-7ebabe66-1f7a-478f-91d4-a93ff03eea46. If it's required to delete it under any c
ircumstances, please set spark.sql.streaming.forceDeleteTempCheckpointLocation to true. Important to know deleting t
emp checkpoint folder is best effort.
-------------------------------------------
Batch: 0
-------------------------------------------
+----+-----+
|word|count|
+----+-----+
+----+-----+

-------------------------------------------
Batch: 1
-------------------------------------------
+--------+-----+
|    word|count|
+--------+-----+
|Learning|    1|
|    well|    1|
+--------+-----+
```

Input console – Producer.
#/opt/kafka/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic logs

```
[root@spark0 learning]# /opt/kafka/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic logs
>Learning well
>kafka learning well
>kafka learning
>
```

Output on the console.



Refer:
https://databricks.com/blog/2017/04/26/processing-data-in-apache-kafka-with-structured-streaming-in-apache-spark-2-2.html

---------- Lab Ends Here ----------

23. Processing Multiple Batches of Streaming Data – 60 Minutes

This Lab depends on the consuming kafka message.

Start the kafka broker.

In this section, we will determine request count by IP using checkpoint concept.

Start the spark shell
spark-shell --packages org.apache.spark:spark-streaming-kafka-0-10_2.12:3.0.1

```
import org.apache.spark.SparkContext
import org.apache.spark.streaming.StreamingContext
import org.apache.spark.streaming.Seconds

import org.apache.kafka.clients.consumer.ConsumerRecord
import org.apache.kafka.common.serialization.StringDeserializer
import org.apache.spark.streaming.kafka010._
import org.apache.spark.streaming.kafka010.LocationStrategies.PreferConsistent
import org.apache.spark.streaming.kafka010.ConsumerStrategies.Subscribe

val kafkaParams = Map[String, Object](
  "bootstrap.servers" -> "localhost:9092",
```

```
  "key.deserializer" -> classOf[StringDeserializer],
  "value.deserializer" -> classOf[StringDeserializer],
  "group.id" -> "use_a_separate_group_id_for_each_stream",
  "auto.offset.reset" -> "latest",
  "enable.auto.commit" -> (false: java.lang.Boolean)
)

val ssc = new StreamingContext(sc,  Seconds(6))

val topics = Array("log")

val stream = KafkaUtils.createDirectStream[String, String](
  ssc,
  PreferConsistent,
  Subscribe[String, String](topics, kafkaParams)
)

val lines = stream.map(_.value)
val userreqs = lines.map(line => (line.split(' ')(0),1)).reduceByKey((x,y) => x+y)


#Total Request By IP Count - Update Function

def updateCount = (newCounts: Seq[Int] ,  state: Option[Int] ) => {
        val newCount = newCounts.foldLeft(0)(_+_)
```

```
        val previousCount = state.getOrElse(0)
        Some(newCount + previousCount)
}

ssc.checkpoint("checkpoints")
val totalUserreqs = userreqs.updateStateByKey(updateCount)
totalUserreqs.print

ssc.start
ssc.awaitTermination
```

```
updateCount: (Seq[Int], Option[Int]) => Some[Int]

scala> ssc.checkpoint("checkpoints")

scala> val totalUserreqs = userreqs.updateStateByKey(updateCount)
totalUserreqs: org.apache.spark.streaming.dstream.DStream[(String, Int)] = org.apache.spark.streaming.dstream.StateDStream@2b834e4f

scala> totalUserreqs.print

scala> ssc.start

-------------------------------------------
Time: 1605521376000 ms
-------------------------------------------

-------------------------------------------
Time: 1605521382000 ms
-------------------------------------------

-------------------------------------------
Time: 1605521388000 ms
```

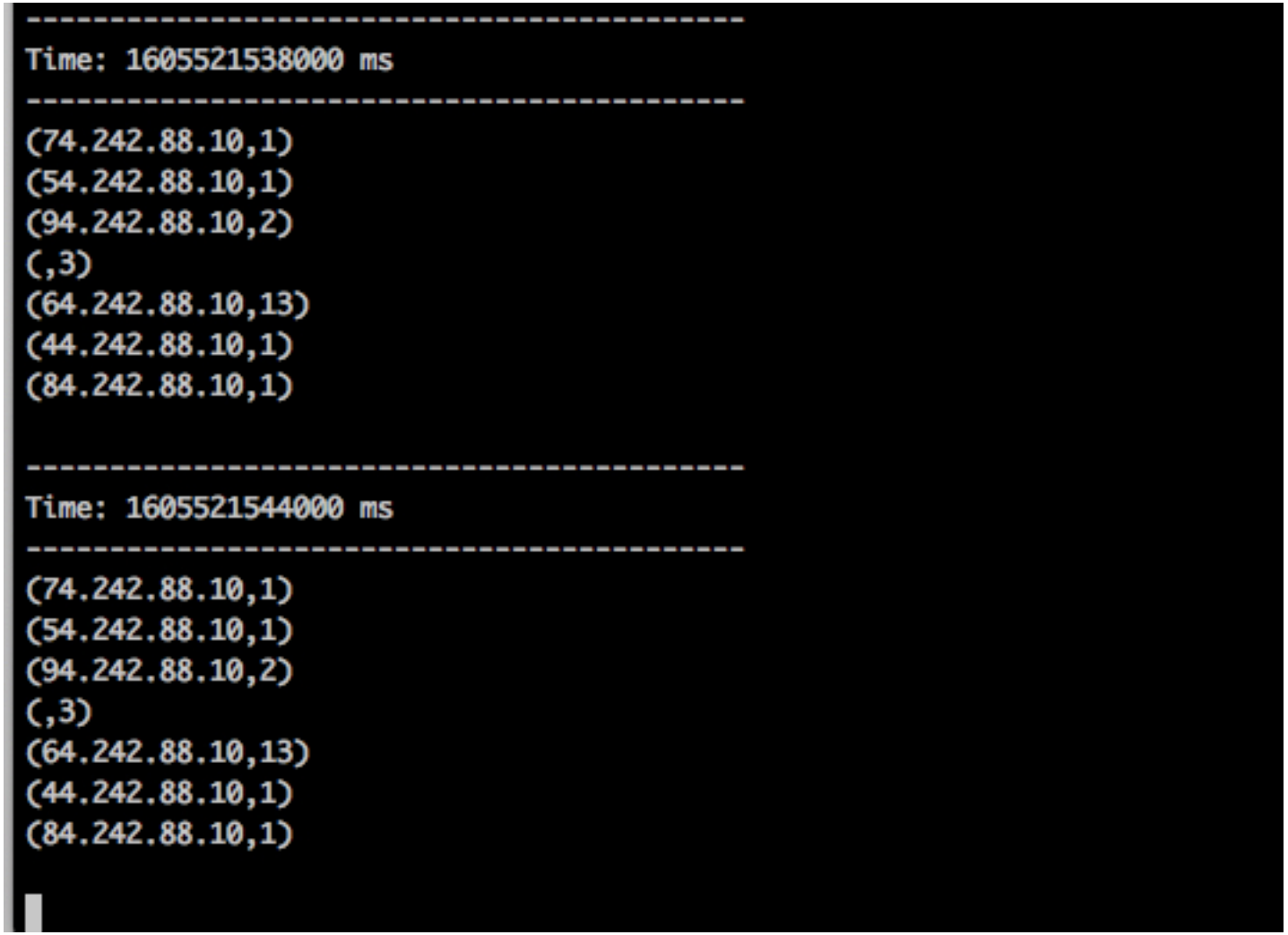
Using the producer console. Send the following messages in two batches. (10 Records at a time.)
#/opt/kafka/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic log

First Batch:
64.242.88.10 - - [07/Mar/2004:16:05:49 -0800] "GET /twiki/bin/edit/Main/Double_bounce_sender?topicparent=Main.ConfigurationVariables HTTP/1.1" 401 12846
64.242.88.10 - - [07/Mar/2004:16:06:51 -0800] "GET /twiki/bin/rdiff/TWiki/NewUserTemplate?rev1=1.3&rev2=1.2 HTTP/1.1" 200 4523
64.242.88.10 - - [07/Mar/2004:16:10:02 -0800] "GET /mailman/listinfo/hsdivision HTTP/1.1" 200 6291
74.242.88.10 - - [07/Mar/2004:16:11:58 -0800] "GET /twiki/bin/view/TWiki/WikiSyntax HTTP/1.1" 200 7352
64.242.88.10 - - [07/Mar/2004:16:20:55 -0800] "GET /twiki/bin/view/Main/DCCAndPostFix HTTP/1.1" 200 5253
84.242.88.10 - - [07/Mar/2004:16:23:12 -0800] "GET /twiki/bin/oops/TWiki/AppendixFileSystem?template=oopsmore&param1=1.12&param2=1.12 HTTP/1.1" 200 11382
64.242.88.10 - - [07/Mar/2004:16:24:16 -0800] "GET /twiki/bin/view/Main/PeterThoeny HTTP/1.1" 200 4924
64.242.88.10 - - [07/Mar/2004:16:29:16 -0800] "GET /twiki/bin/edit/Main/Header_checks?topicparent=Main.ConfigurationVariables HTTP/1.1" 401 12851

Second Batch
94.242.88.10 - - [07/Mar/2004:16:30:29 -0800] "GET /twiki/bin/attach/Main/OfficeLocations HTTP/1.1" 401 12851
64.242.88.10 - - [07/Mar/2004:16:31:48 -0800] "GET /twiki/bin/view/TWiki/WebTopicEditTemplate HTTP/1.1" 200 3732
64.242.88.10 - - [07/Mar/2004:16:32:50 -0800] "GET /twiki/bin/view/Main/WebChanges HTTP/1.1" 200 40520
44.242.88.10 - - [07/Mar/2004:16:33:53 -0800] "GET /twiki/bin/edit/Main/Smtpd_etrn_restrictions?topicparent=Main.ConfigurationVariables HTTP/1.1" 401 12851
64.242.88.10 - - [07/Mar/2004:16:35:19 -0800] "GET /mailman/listinfo/business HTTP/1.1" 200 6379
64.242.88.10 - - [07/Mar/2004:16:36:22 -0800] "GET /twiki/bin/rdiff/Main/WebIndex?rev1=1.2&rev2=1.1 HTTP/1.1" 200 46373
54.242.88.10 - - [07/Mar/2004:16:37:27 -0800] "GET /twiki/bin/view/TWiki/DontNotify HTTP/1.1" 200 4140
64.242.88.10 - - [07/Mar/2004:16:39:24 -0800] "GET /twiki/bin/view/Main/TokyoOffice HTTP/1.1" 200 3853
64.242.88.10 - - [07/Mar/2004:16:43:54 -0800] "GET /twiki/bin/view/Main/MikeMannix HTTP/1.1" 200 3686

```
>
>
>64.242.88.10 - - [07/Mar/2004:16:05:49 -0800] "GET /twiki/bin/edit/Main/Double_bounce_sender?topicparent=Main.ConfigurationVariables HTTP/1.1" 4
01 12846
64.242.88.10 - - [07/Mar/2004:16:06:51 -0800] "GET /twiki/bin/rdiff/TWiki/NewUserTemplate?rev1=1.3&rev2=1.2 HTTP/1.1" 200 4523
64.242.88.10 - - [07/Mar/2004:16:10:02 -0800] "GET /mailman/listinfo/hsdivision HTTP/1.1" 200 6291
74.242.88.10 - - [07/Mar/2004:16:11:58 -0800] "GET /twiki/bin/view/TWiki/WikiSyntax HTTP/1.1" 200 7352
64.242.88.10 - - [07/Mar/2004:16:20:55 -0800] "GET /twiki/bin/view/Main/DCCAndPostFix HTTP/1.1" 200 5253
84.242.88.10 - - [07/Mar/2004:16:23:12 -0800] "GET /twiki/bin/oops/TWiki/AppendixFileSystem?template=oopsmore&param1=1.12&param2=1.12 HTTP/1.1" 2
00 11382
64.242.88.10 - - [07/Mar/20>>04:16:24:16 -0800] "GET /twiki/bin/view/Main/PeterThoeny HTTP/1.1" 200 4924
64.242.88.10 - - [07/Mar/2004:16:29:16 -0800] "GET /twiki/bin/edit/Main/Header_checks?topicparent=Main.ConfigurationVariables HTTP/1.1" 401 12851
94.242.88.10 - - [07/Mar/2004:16:30:29 -0800] "GET /twiki/bin/attach/Main/OfficeLocations HTTP/1.1" 401 12851
64.242.88.10 - - [07/Mar/2004:16:31:48 -0800] "GET /twiki/bin/view/TWiki/WebTopicEditTemplate HTTP/1.1" 200 3732
64.242.88.10 - - [07/Mar/2004:16:32:50 -0800] "GET /twiki/bin/view/Main/WebChanges HTTP/1.1" 200 40520
44.242.88.10 - - [07/Mar/2004:16:33:53 -0800] "GET /twiki/bin/edit/Main/Smtpd_etrn_restrictions?topicparent=Main.ConfigurationVariables HTTP/1.1"
 401 12851
64.242.88.10 - - [07/Mar/2004:16:35:19 -0800] "GET /mailman/list>>>>info/business HTTP/1.1" 200 6379
64.242.88.10 - - [07/Mar/2004:16:36:22 -0800] "GET /twiki/bin/rdiff/Main/WebIndex?rev1=1.2&rev2=1.1 HTTP/1.1" 200 46373
54.242.88.10 - - [07/Mar/2004:16:37:27 -0800] "GET /twiki/bin/view/TWiki/DontNotify HTTP/1.1" 200 4140
64.242.88.10 - - [07/Mar/2004:16:39:24 -0800] "GET /twiki/bin/view/Main/TokyoOffice HTTP/1.1" 200 3853
64.242.88.10 - - [07/Mar/2004:16:43:54 -0800] "GET /twiki/bin/view/Main/MikeMannix HTTP/1.1" 200 3686
94.242.88.10 - - [07/Mar/2004:16:45:56 -0800] "GET /twiki/bin/attach/Main/PostfixCommands HTTP/1.1" 401 12846
64.242.88.10 - - [07/Mar/2004:16:47:12 -0800] "GET /robots.txt HTTP/1.1" 200 68
>>>>>>>>>>>>>
```

Verify the output in the spark-shell console: As you can see the count is being displayed for each IP continuously.



Count and Sort IP Requests by Window. – Open a new console.

```
# spark-shell --packages org.apache.spark:spark-streaming-kafka-0-10_2.12:3.0.1


import org.apache.spark.SparkContext
import org.apache.spark.streaming.StreamingContext
import org.apache.spark.streaming.Seconds

import org.apache.kafka.clients.consumer.ConsumerRecord
import org.apache.kafka.common.serialization.StringDeserializer
import org.apache.spark.streaming.kafka010._
import org.apache.spark.streaming.kafka010.LocationStrategies.PreferConsistent
import org.apache.spark.streaming.kafka010.ConsumerStrategies.Subscribe


val kafkaParams = Map[String, Object](
  "bootstrap.servers" -> "localhost:9092",
  "key.deserializer" -> classOf[StringDeserializer],
  "value.deserializer" -> classOf[StringDeserializer],
  "group.id" -> "use_a_separate_group_id_for_each_stream",
  "auto.offset.reset" -> "latest",
  "enable.auto.commit" -> (false: java.lang.Boolean)
```

```
)

val ssc = new StreamingContext(sc,  Seconds(2))


val topics = Array("log")


val stream = KafkaUtils.createDirectStream[String, String](
  ssc,
  PreferConsistent,
  Subscribe[String, String](topics, kafkaParams)
)

 import org.apache.spark.streaming.{Seconds, StreamingContext, Time, Minutes}


val logs = stream.map(_.value)

val reqcountsByWindow = logs.map(line => (line.split('
')(0),1)).reduceByKeyAndWindow((v1: Int, v2: Int) => v1+v2,Minutes(5),Seconds(30))
```

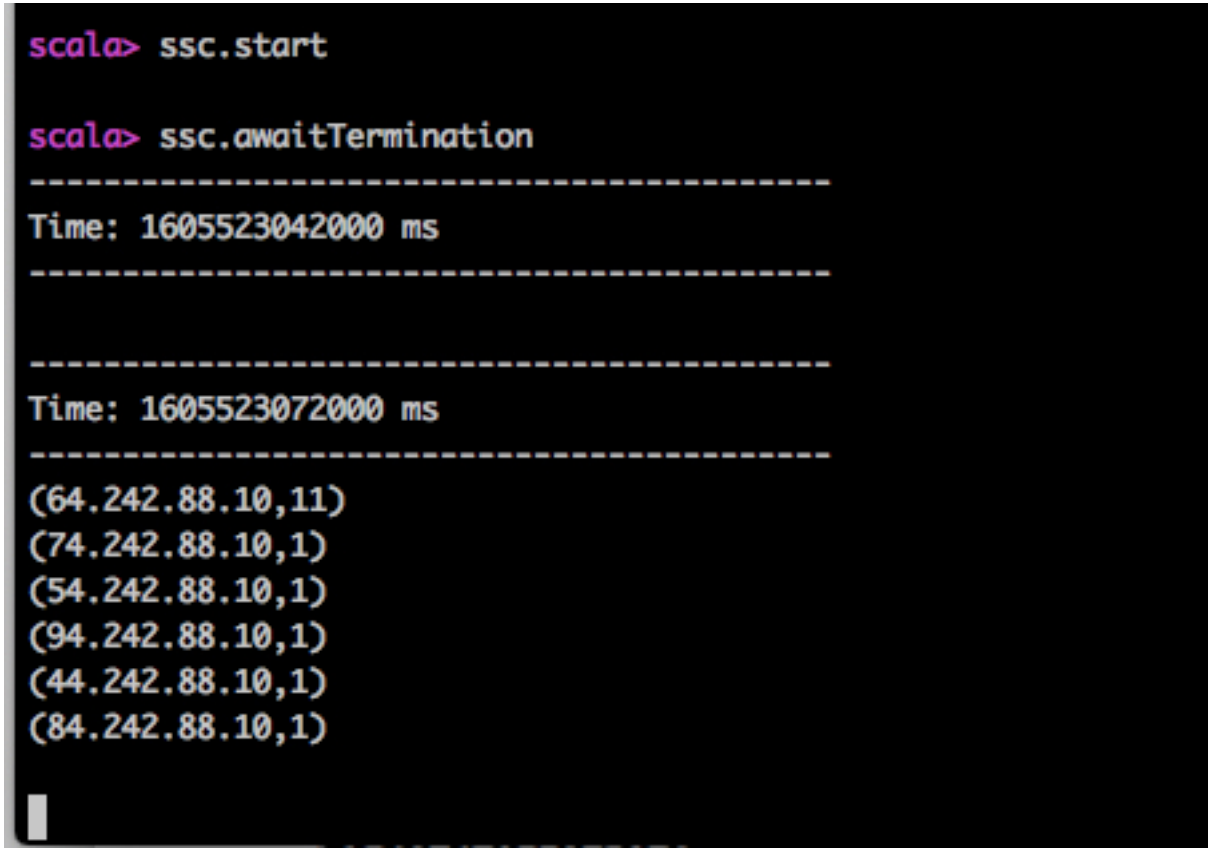
```
val topreqsByWindow=reqcountsByWindow.map(pair => pair.swap).transform(rdd =>
rdd.sortByKey(false))

topreqsByWindow.map(pair => pair.swap).print


ssc.start

ssc.awaitTermination
```

Input in the Producer CLI

/opt/kafka/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic log

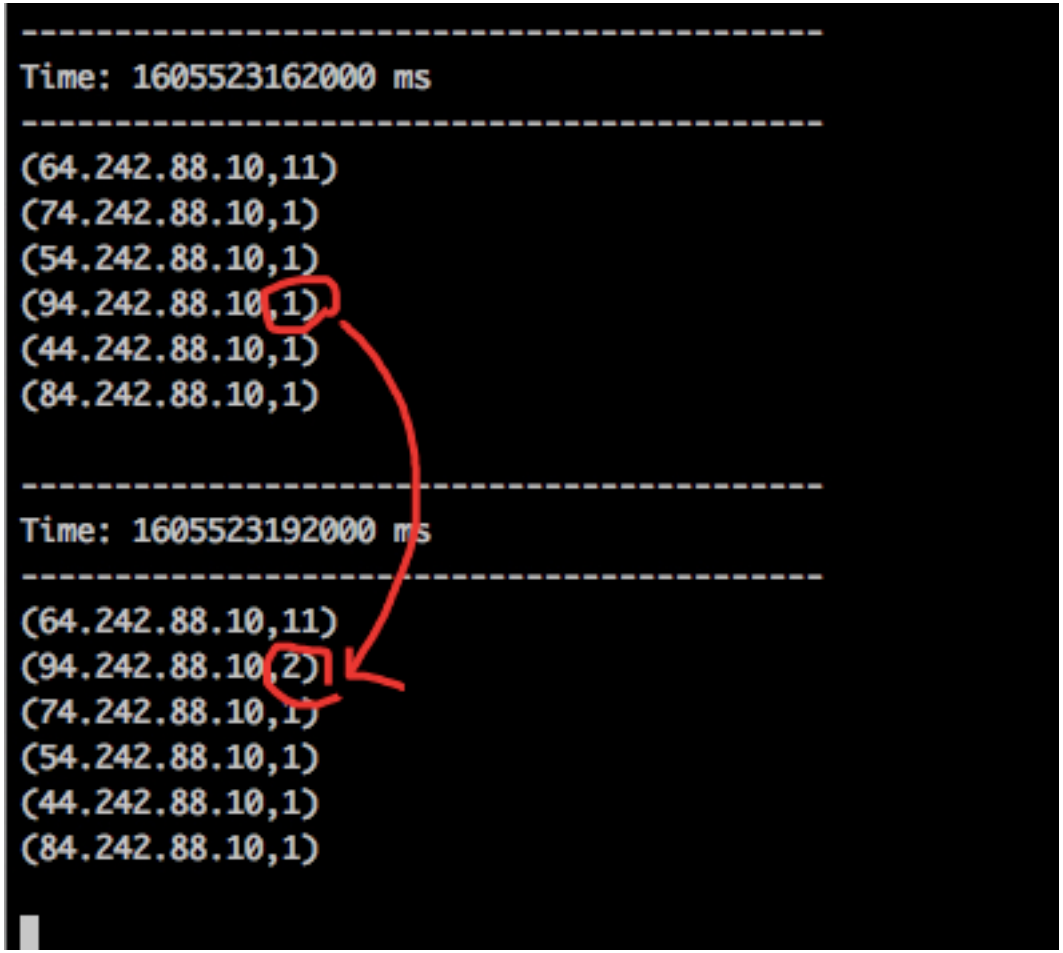
64.242.88.10 - - [07/Mar/2004:16:05:49 -0800] "GET /twiki/bin/edit/Main/Double_bounce_sender?topicparent=Main.ConfigurationVariables HTTP/1.1" 401 12846
64.242.88.10 - - [07/Mar/2004:16:06:51 -0800] "GET /twiki/bin/rdiff/TWiki/NewUserTemplate?rev1=1.3&rev2=1.2 HTTP/1.1" 200 4523
64.242.88.10 - - [07/Mar/2004:16:10:02 -0800] "GET /mailman/listinfo/hsdivision HTTP/1.1" 200 6291
74.242.88.10 - - [07/Mar/2004:16:11:58 -0800] "GET /twiki/bin/view/TWiki/WikiSyntax HTTP/1.1" 200 7352
64.242.88.10 - - [07/Mar/2004:16:20:55 -0800] "GET /twiki/bin/view/Main/DCCAndPostFix HTTP/1.1" 200 5253
84.242.88.10 - - [07/Mar/2004:16:23:12 -0800] "GET /twiki/bin/oops/TWiki/AppendixFileSystem?template=oopsmore&param1=1.12&param2=1.12 HTTP/1.1" 200 11382
64.242.88.10 - - [07/Mar/2004:16:24:16 -0800] "GET /twiki/bin/view/Main/PeterThoeny HTTP/1.1" 200 4924
64.242.88.10 - - [07/Mar/2004:16:29:16 -0800] "GET /twiki/bin/edit/Main/Header_checks?topicparent=Main.ConfigurationVariables HTTP/1.1" 401 12851
94.242.88.10 - - [07/Mar/2004:16:30:29 -0800] "GET /twiki/bin/attach/Main/OfficeLocations HTTP/1.1" 401 12851
64.242.88.10 - - [07/Mar/2004:16:31:48 -0800] "GET /twiki/bin/view/TWiki/WebTopicEditTemplate HTTP/1.1" 200 3732
64.242.88.10 - - [07/Mar/2004:16:32:50 -0800] "GET /twiki/bin/view/Main/WebChanges HTTP/1.1" 200 40520
44.242.88.10 - - [07/Mar/2004:16:33:53 -0800] "GET /twiki/bin/edit/Main/Smtpd_etrn_restrictions?topicparent=Main.ConfigurationVariables HTTP/1.1" 401 12851
64.242.88.10 - - [07/Mar/2004:16:35:19 -0800] "GET /mailman/listinfo/business HTTP/1.1" 200 6379
64.242.88.10 - - [07/Mar/2004:16:36:22 -0800] "GET /twiki/bin/rdiff/Main/WebIndex?rev1=1.2&rev2=1.1 HTTP/1.1" 200 46373
54.242.88.10 - - [07/Mar/2004:16:37:27 -0800] "GET /twiki/bin/view/TWiki/DontNotify HTTP/1.1" 200 4140
64.242.88.10 - - [07/Mar/2004:16:39:24 -0800] "GET /twiki/bin/view/Main/TokyoOffice HTTP/1.1" 200 3853
64.242.88.10 - - [07/Mar/2004:16:43:54 -0800] "GET /twiki/bin/view/Main/MikeMannix HTTP/1.1" 200 3686
94.242.88.10 - - [07/Mar/2004:16:45:56 -0800] "GET /twiki/bin/attach/Main/PostfixCommands HTTP/1.1" 401 12846

On spark Console:

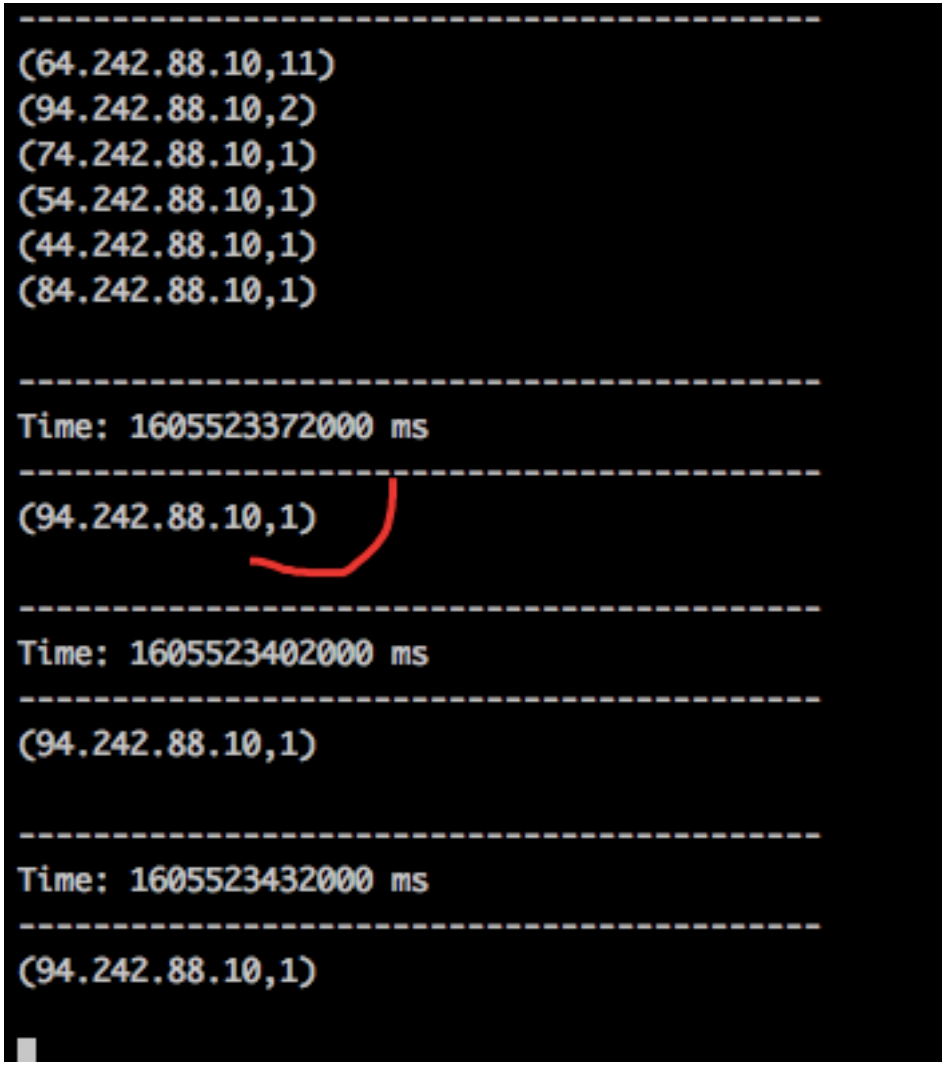


Send again the following: After some interval > 30 seconds

94.242.88.10 - - [07/Mar/2004:16:47:12 -0800] "GET /robots.txt HTTP/1.1" 200 68



As shown above the count is increased by 1 in the next interval.
After 5 mins it should be as shown below:

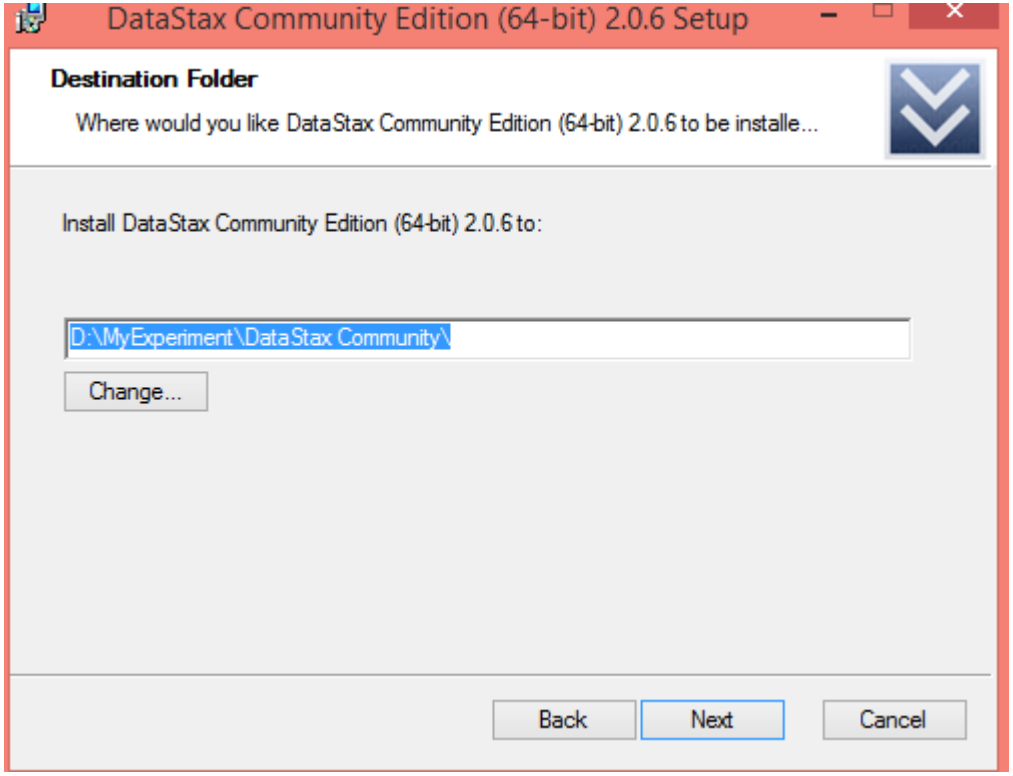


It slides by 5 mins, hence shown the last 1 changes in that period.

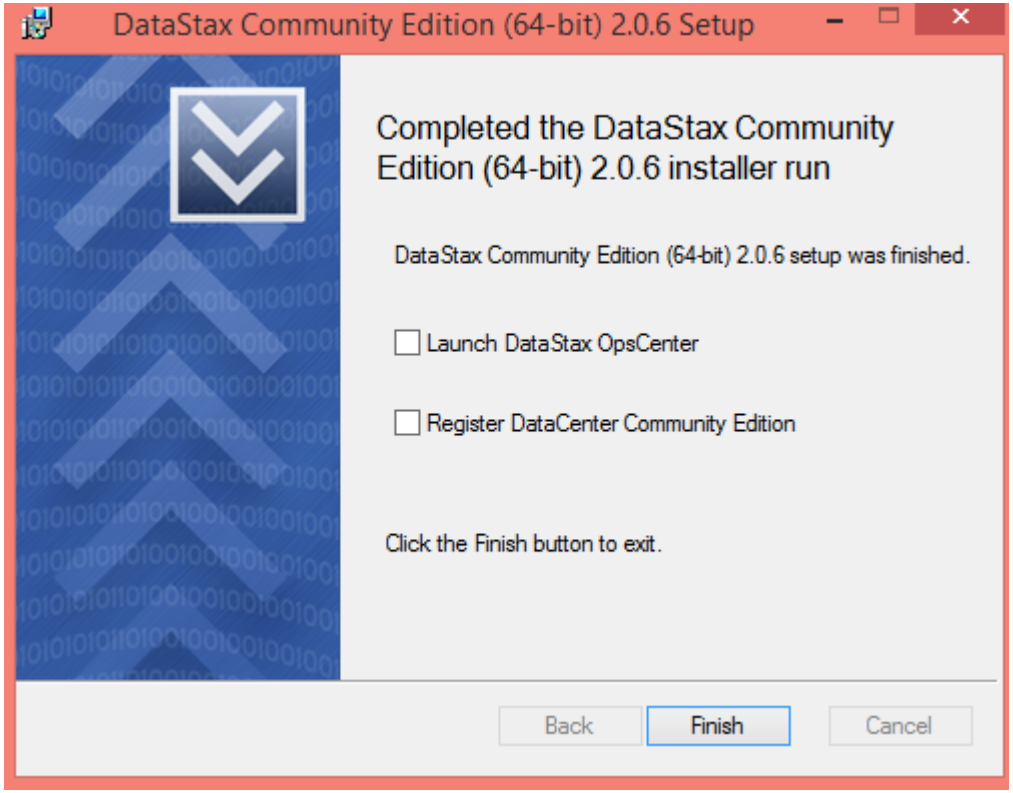
---------- Lab Ends Here ----------

24. Exercise : Spark Integration With Cassandra

Install Cassandra library for using as a client from your laptop.
[Optional, If you are using windows server only]Install cassandra. double click datastax-community-64bit_2.0.6.exe. You can leave all the setting default. In my case I am changing the installation folder

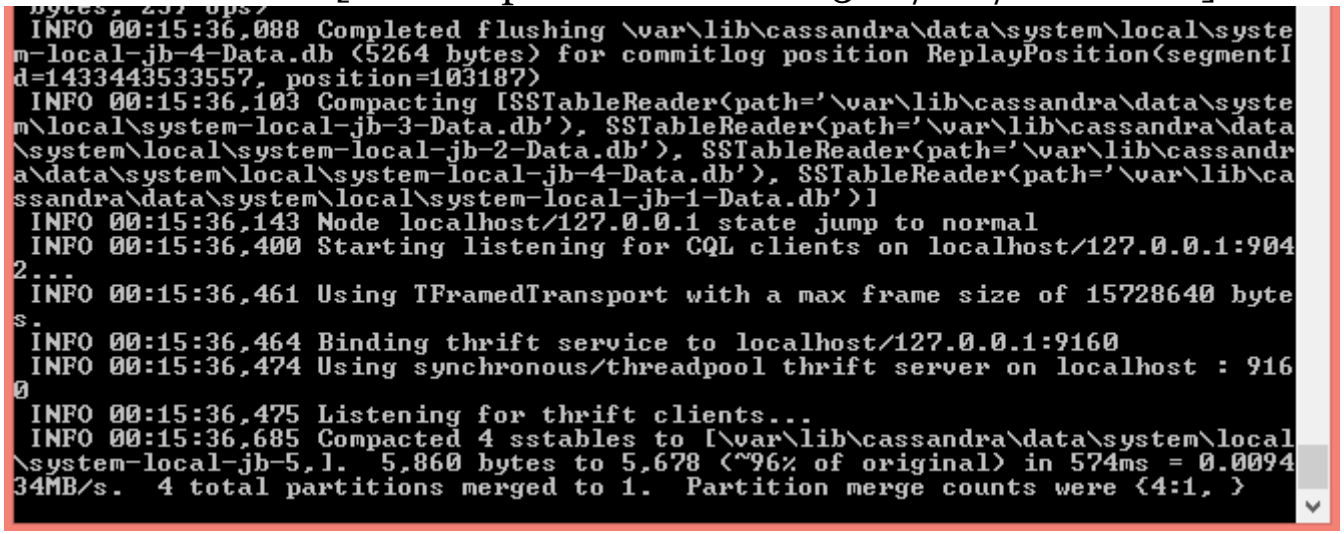


Click Finish with all default value and wait till it get installed.



Finish to complete the installation.

You need to have cassandra cluster before proceeding ahead:
Refer Tutorial: Installation of Cassandra. If you don't have this document, kindly ask the trainer. Else, you can use the slave node as a Cassandra cluster. Ensure to start the Cassandra cluster[hints : apache-cassandra-3.10/bin/cassandra].



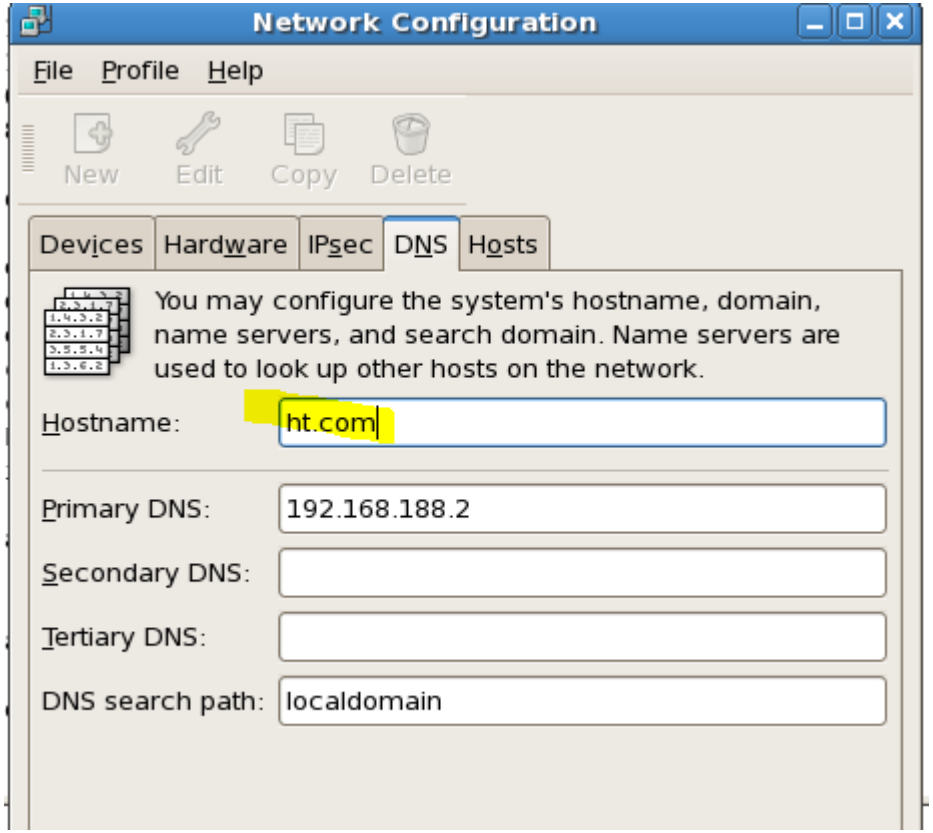
Configuring Spark
// You can find this file in sbin folder of spark installation folder
cp spark-defaults.conf.template spark-defaults.conf
Update the config as follows:

```
[root@localhost sbin]# pwd
/spark/spark-1.3.0-bin-hadoop2.4/sbin
[root@localhost sbin]# cd ../
[root@localhost spark-1.3.0-bin-hadoop2.4]# cd conf
[root@localhost conf]# ls
fairscheduler.xml.template   slaves.template
log4j.properties.template    spark-defaults.conf.template
metrics.properties.template  spark-env.sh.template
[root@localhost conf]# cp spark-defaults.conf.template spark-defaults.conf
```

hostname should be as follow: or noted down your hostname

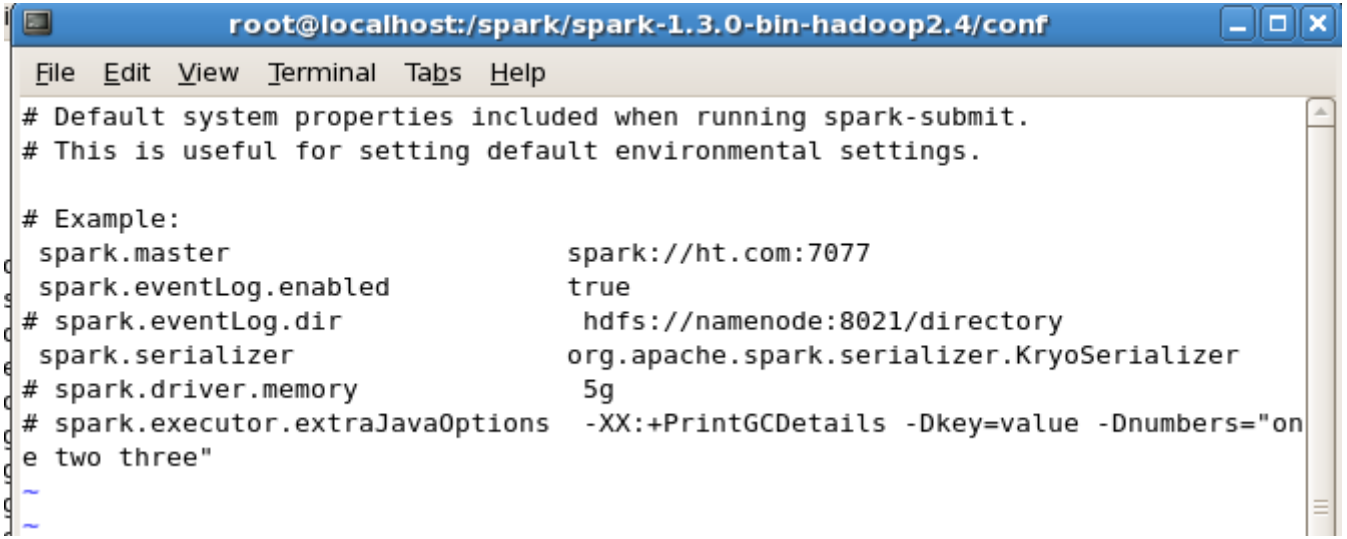
```
[root@localhost ~]# hostname
ht.com
[root@localhost ~]#
```

You can set the hostname as follows:

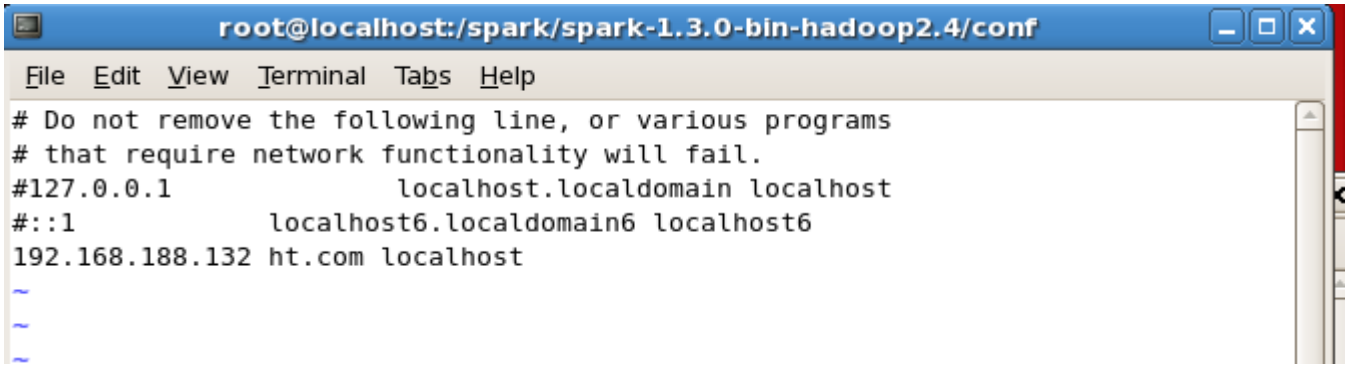


Ensure to restart the network after changing the hostname:
service network restart

Update the spark-defaults.conf as follows using vi editor. Ensure to enter the hostname in the spark master value.



Ensure to map the hostname with the ip , verify the ip of your machine using ifconfig command



Let us start one master and a worker as follows: Open two terminal and execute as follows
cd /spark/spark-2.1
sh sbin/start-master.sh

sh sbin/start-slave.sh spark://ht.com:7077

verify log to confirm the start up as follows: You can find both the long inside this folder (/spark/spark-1.3.0-bin-hadoop2.4/logs) [Specify IP instead of hostname if unable to connect or start up]

```
[root@ht bin]# cd /spark/spark-1.3.0-bin-hadoop2.4
[root@ht spark-1.3.0-bin-hadoop2.4]# sh sbin/start-master.sh 1
starting org.apache.spark.deploy.master.Master, logging to /spark/spark-1.3.0-bi
n-hadoop2.4/sbin/../logs/spark-root-org.apache.spark.deploy.master.Master-1-ht.c
om.out
[root@ht spark-1.3.0-bin-hadoop2.4]# sh sbin/start-slave.sh 1 spark://ht.com:70
77
starting org.apache.spark.deploy.worker.Worker, logging to /spark/spark-1.3.0-bi
n-hadoop2.4/sbin/../logs/spark-root-org.apache.spark.deploy.worker.Worker-1-ht.c
om.out
[root@ht spark-1.3.0-bin-hadoop2.4]# jps
13010 Worker
12341 CassandraDaemon
13095 Jps
12892 Master
[root@ht spark-1.3.0-bin-hadoop2.4]#
```

master log:
more spark-root-org.apache.spark.deploy.master.Master-1-localhost.localdomain.out

```
15/06/07 00:33:04 INFO AbstractConnector: Started SelectChannelConnector@0.0.0.0
:8080
15/06/07 00:33:04 INFO Utils: Successfully started service 'MasterUI' on port 80
80.
15/06/07 00:33:04 INFO MasterWebUI: Started MasterWebUI at http://ht.com:8080
15/06/07 00:33:04 INFO Master: I have been elected leader! New state: ALIVE
15/06/07 00:35:16 INFO Master: Registering worker ht.com:58645 with 1 cores, 512
.0 MB RAM
[root@ht logs]#
```

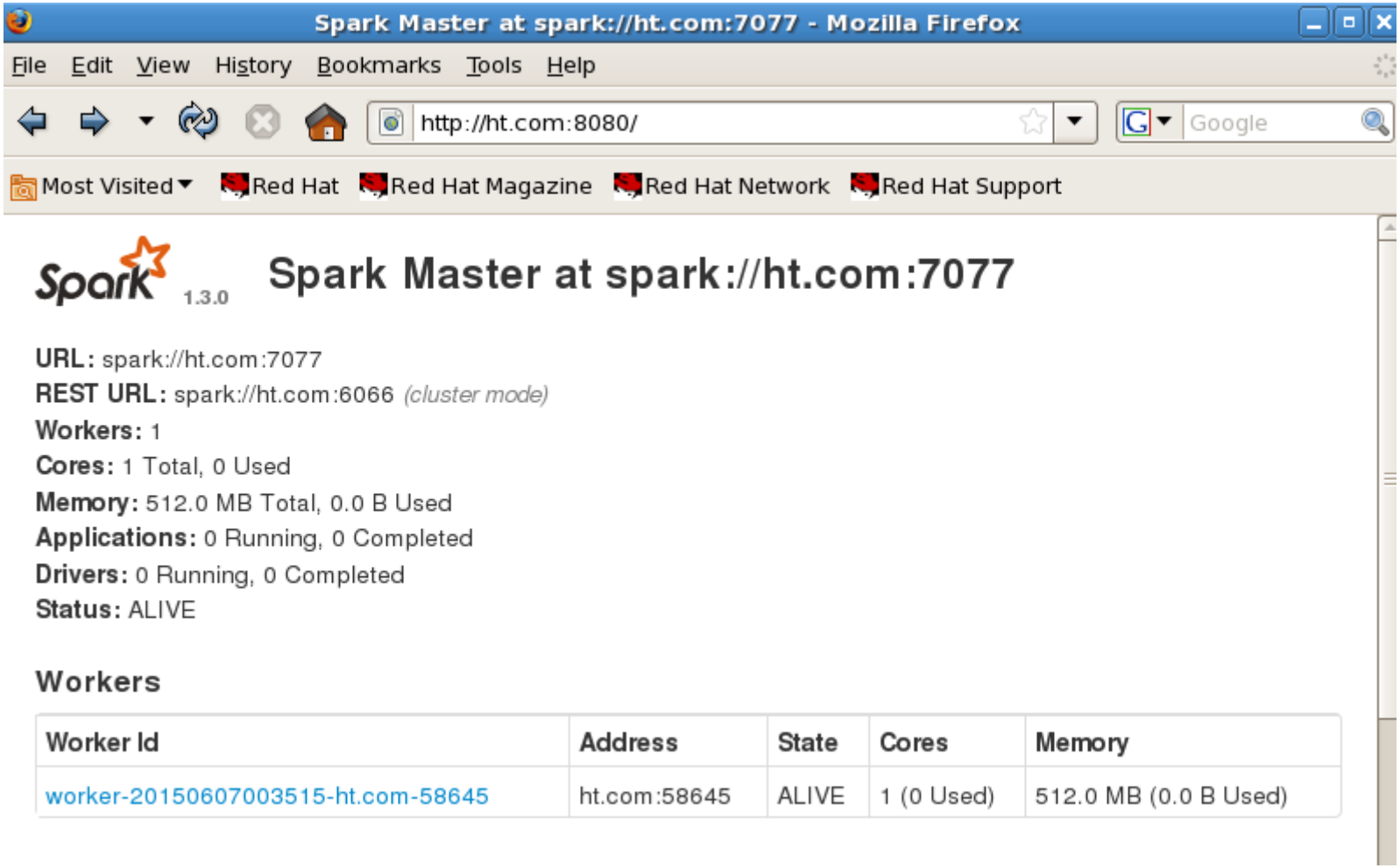
Worker log :

more spark-root-org.apache.spark.deploy.worker.Worker-1-localhost.localdomain.out

```
:8081
15/06/07 00:35:15 INFO Utils: Successfully started service 'WorkerUI' on port 80
81.
15/06/07 00:35:15 INFO WorkerWebUI: Started WorkerWebUI at http://ht.com:8081
15/06/07 00:35:15 INFO Worker: Connecting to master akka.tcp://sparkMaster@ht.co
m:7077/user/Master...
15/06/07 00:35:16 INFO Worker: Successfully registered with master spark://ht.co
m:7077
```

browse the Spark master webui.

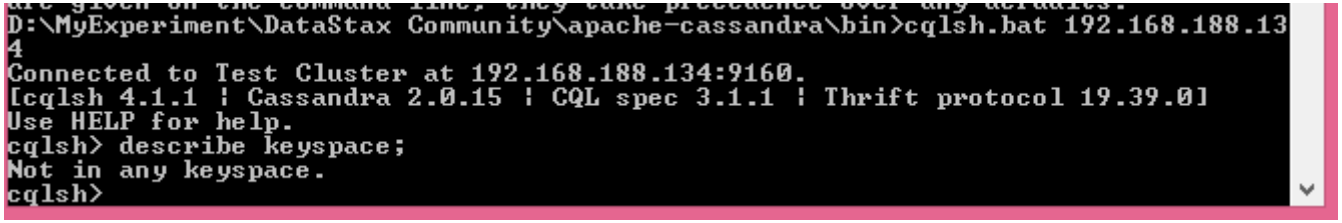
http://ht.com:8080/ or http://192.168.188.134:8080/


Spark Master at spark://ht.com:7077 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://ht.com:8080/
Google
Most Visited
Red Hat
Red Hat Magazine
Red Hat Network
Red Hat Support
Spark 1.3.0
Spark Master at spark://ht.com:7077
URL: spark://ht.com:7077
REST URL: spark://ht.com:6066 (cluster mode)
Workers: 1
Cores: 1 Total, 0 Used
Memory: 512.0 MB Total, 0.0 B Used
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE
Workers
Worker Id
Address
State
Cores
Memory
worker-20150607003515-ht.com-58645
ht.com:58645
ALIVE
1 (0 Used)
512.0 MB (0.0 B Used)

Ensure that your cassandra custer is running.

Configure client on your windows client machine.

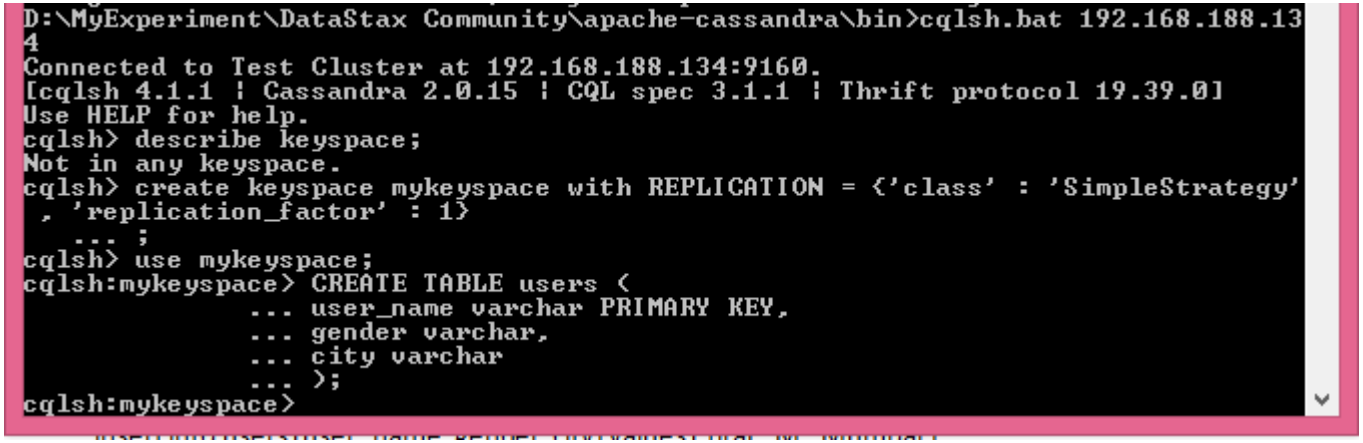
Install python in your host machine: python-2.7.6.msi
Include python installation in the PATH variable.
Extract Cassandra in your client machine that is Host. (Window) This will be your client to connect to Cassandra server hosted in Linux.
Create key space as follow:
Connect to Cassandra server from window client. You need to your cassandra installation folder
cqlsh.bat 192.168.188.134



create keyspace mykeyspace with REPLICATION = {'class' : 'SimpleStrategy' ,
'replication_factor' : 1};
use mykeyspace;

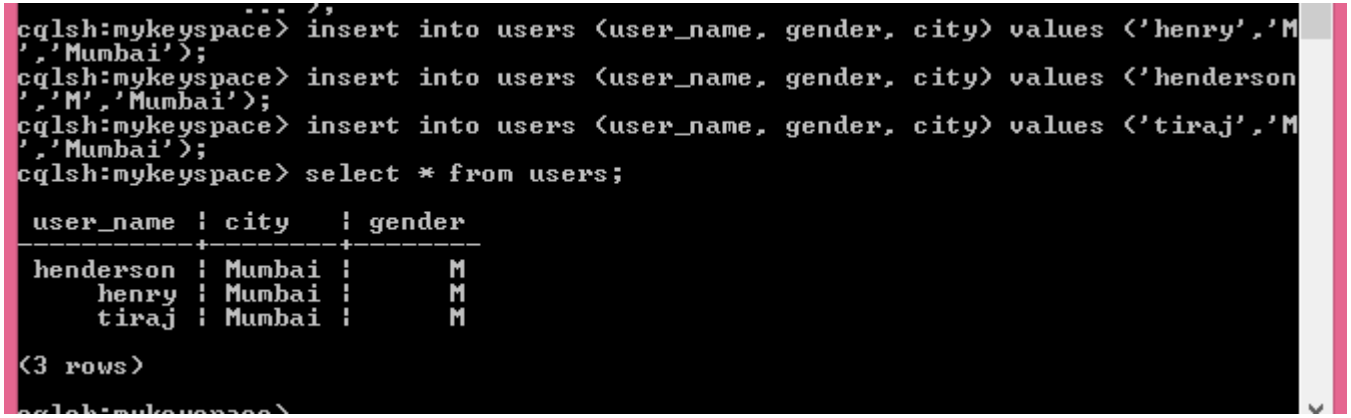
CREATE TABLE users (
user_name varchar PRIMARY KEY,
gender varchar,

city varchar
);



insert 3 records as follow:

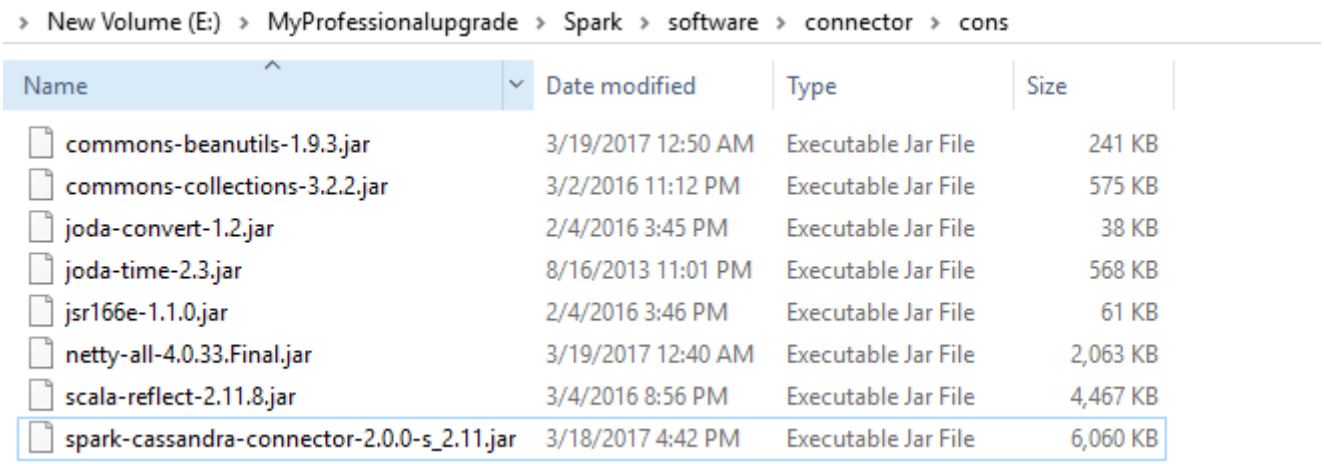
insert into users (user_name, gender, city) values ('henry','M','Mumbai');
insert into users (user_name, gender, city) values ('henderson','M','Mumbai');
insert into users (user_name, gender, city) values ('tiraj','M','Mumbai');



Create a temporary folder which is required for Cassandra connector:
mkdir /tmp/spark-events

Start the spark cluster as follows: If you have shutdown else go to next step
sh sbin/start-master.sh
sh sbin/start-slave.sh spark://ht.com:7077

Copy cassandra connector jars in spark home /lib folder of the VM - Spark Master.



Open a console in your Spark VM and go to the spark home folder then start spark shell by specifying the host name of cassandra cluster and the spark master.

#cd {Spark - Home}\

#bin\spark-shell --conf spark.cassandra.connection.host=192.168.188.174 --master spark://192.168.188.178:7077

```
[root@master bin]# spark-shell --conf spark.cassandra.connection.host=192.168.188.174 --master spark://192.168.188.178:7077
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
17/03/19 01:30:05 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
17/03/19 01:30:29 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
Spark context Web UI available at http://192.168.188.178:4040
Spark context available as 'sc' (master = spark://192.168.188.178:7077, app id = app-20170319013010-0022).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.1.0
      /_/

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_121)
Type in expressions to have them evaluated.
Type :help for more information.
```

Execute the following command to retrieve data from cassandra using spark console.

#import com.datastax.spark.connector._
#import org.apache.spark.SparkConf
#import org.apache.spark.SparkContext

#val rdd = sc.cassandraTable("mykeyspace", "users")

// You need to wait for a moment till you get the object details being displayed as above before submitting another command.
println(rdd.count)

```
scala> import com.datastax.spark.connector._
import com.datastax.spark.connector._

scala> import org.apache.spark.SparkConf
import org.apache.spark.SparkConf

scala> import org.apache.spark.SparkContext
import org.apache.spark.SparkContext

scala> val rdd = sc.cassandraTable("mykeyspace", "users")
rdd: com.datastax.spark.connector.rdd.CassandraTableScanRDD[com.datastax.spark.connector.CassandraRow] = CassandraTableScanRDD[0] at RDD at CassandraRDD.sca
la:19

scala> println(rdd.count)
4
```

println(rdd.first)

```
scala> println(rdd.first)
15/06/09 23:11:53 INFO Cluster: New Cassandra host ht.com/192.168.188.134:9042 a
dded
15/06/09 23:11:53 INFO CassandraConnector: Connected to Cassandra cluster: Test
Cluster
15/06/09 23:11:53 INFO SparkContext: Starting job: take at CassandraRDD.scala:11
8
15/06/09 23:11:53 INFO TaskSetManager: Starting task 0.0 in stage 1.0 (TID 1, ht
.com, NODE_LOCAL, 29389 bytes)
15/06/09 23:11:53 INFO BlockManagerInfo: Added broadcast_1_piece0 in memory on h
t.com:42351 (size: 2.8 KB, free: 267.3 MB)
15/06/09 23:11:53 INFO CassandraConnector: Disconnected from Cassandra cluster:
Test Cluster
15/06/09 23:11:53 INFO DAGScheduler: Stage 1 (take at CassandraRDD.scala:118) fi
nished in 0.468 s
15/06/09 23:11:53 INFO DAGScheduler: Job 1 finished: take at CassandraRDD.scala:
118, took 0.531818 s
CassandraRow{user_name: henderson, city: Mumbai, gender: M}
```

val collection = sc.parallelize(Seq(("rajnita", "Mumbai","F")))

```
scala> val collection = sc.parallelize(Seq(("rajnita", "Mumbai","F")))
collection: org.apache.spark.rdd.RDD[(String, String, String)] = ParallelCollect
ionRDD[3] at parallelize at <console>:28
```

collection.saveToCassandra("mykeyspace", "users", SomeColumns("user_name", "city","gender"))

```
scala> collection.saveToCassandra("mykeyspace", "users", SomeColumns("user_name"
, "city","gender"))
15/06/09 23:12:15 INFO Cluster: New Cassandra host /192.168.188.134:9042 added
15/06/09 23:12:15 INFO CassandraConnector: Connected to Cassandra cluster: Test
Cluster
15/06/09 23:12:16 INFO SparkContext: Starting job: runJob at RDDFunctions.scala:
36
15/06/09 23:12:16 INFO TaskSetManager: Finished task 0.0 in stage 2.0 (TID 2) in
 471 ms on ht.com (1/2)
15/06/09 23:12:16 INFO DAGScheduler: Stage 2 (runJob at RDDFunctions.scala:36) f
inished in 0.881 s
15/06/09 23:12:16 INFO DAGScheduler: Job 2 finished: runJob at RDDFunctions.scal
a:36, took 0.901716 s

scala> 15/06/09 23:12:16 INFO TaskSetManager: Finished task 1.0 in stage 2.0 (TI
D 3) in 427 ms on ht.com (2/2)
15/06/09 23:12:16 INFO TaskSchedulerImpl: Removed TaskSet 2.0, whose tasks have
all completed, from pool
```

Verify the user now using CQL client.

select * from users;

```
cqlsh:mykeyspace> select * from users;

 user_name | city    | gender
-----------+---------+--------
 henderson | Mumbai |      M
   rajnita | Mumbai |      F
     henry | Mumbai |      M
     tiraj | Mumbai |      M

(4 rows)

cqlsh:mykeyspace>
```

Congratulations! You have successfully retrieve and insert records in the cassandra cluster using Spark.

25. Scala - Broadcast & Repartition – 45 Minutes

Open a spark shell terminal and perform the following steps. In this we are broadcasting the sates and countries details to all the executor.

Understanding BroadCast.

```
#spark-shell
val states = Map(("NY","New York"),("CA","California"),("FL","Florida"))

val countries = Map(("USA","United States of America"),("IN","India"))

val broadcastStates = spark.sparkContext.broadcast(states)

val broadcastCountries = spark.sparkContext.broadcast(countries)

val data = Seq(("James","Smith","USA","CA"),
  ("Michael","Rose","USA","NY"),
  ("Robert","Williams","USA","CA"),
  ("Maria","Jones","USA","FL")
)

val columns = Seq("firstname","lastname","country","state")

import spark.sqlContext.implicits._
```

```
  val df = data.toDF(columns:_*)
```

```
val df2 = df.map(row=>{
    val country = row.getString(2)
    val state = row.getString(3)
   val fullCountry = broadcastCountries.value.get(country).get
    val fullState = broadcastStates.value.get(state).get
    (row.getString(0),row.getString(1),fullCountry,fullState)
  }).toDF(columns:_*)

  df2.show(false)
```

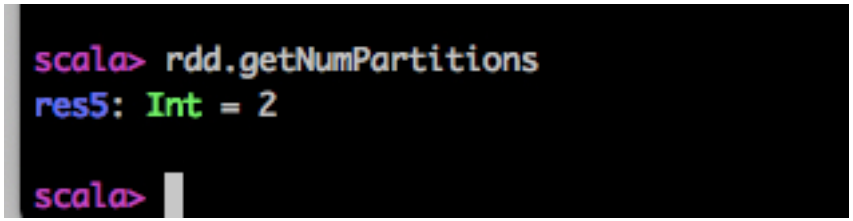
```
scala>   val df = data.toDF(columns:_*)
df: org.apache.spark.sql.DataFrame = [firstname: string, lastname: string ... 2 more fields]

scala>  val df2 = df.map(row=>{
     |     val country = row.getString(2)
     |     val state = row.getString(3)
     |
     |     val fullCountry = broadcastCountries.value.get(country).get
     |     val fullState = broadcastStates.value.get(state).get
     |     (row.getString(0),row.getString(1),fullCountry,fullState)
     |   }).toDF(columns:_*)
df2: org.apache.spark.sql.DataFrame = [firstname: string, lastname: string ... 2 more fields]

scala> df2.show(false)
+---------+--------+----------------------+----------+
|firstname|lastname|country               |state     |
+---------+--------+----------------------+----------+
|James    |Smith   |United States of America|California|
|Michael  |Rose    |United States of America|New York  |
|Robert   |Williams|United States of America|California|
|Maria    |Jones   |United States of America|Florida   |
+---------+--------+----------------------+----------+
```

Understanding Repartition

create a simple RDD using the `sc.parallelize` method to determine the number of partitions spark creates by default.

```
val data = Array(1, 2, 3, 4, 5,6,7,8,9,10,11)
```

```
val rdd = sc.parallelize(data)
rdd.getNumPartitions
```



As shown above, spark creates 2 partitions by default here. Now, you must be wondering where this default number is come from.

Spark has a default parallelism parameter which is determined by, `sc.defaultParallelism`. When we run this method, it returns 2 as shown below,

```
sc.defaultParallelism
```

This is how the data is present inside each partition. Spark uses an internal Hash Partitioning Scheme to split the data into these smaller chunks.

We can use the `rdd.glom` method to display the partitions in a list.

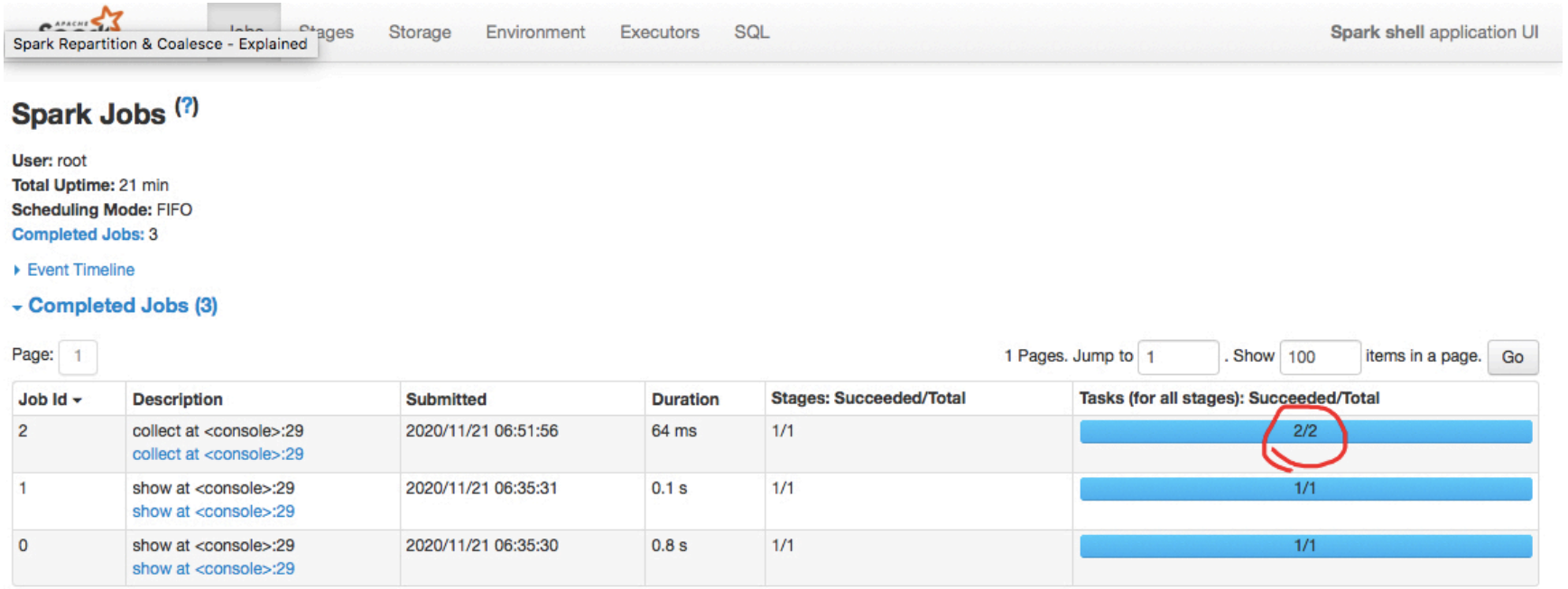
`glom` - returns an RDD having the elements within each partition in a separate list

#rdd.glom.collect

```
scala> sc.defaultParallelism
res6: Int = 2

scala> rdd.glom.co
coalesce    collectAsync    context    count          countApproxDistinct    countByValue
collect     compute         copy       countApprox    countAsync             countByValueApprox

scala> rdd.glom.collect
collect   collectAsync

scala> rdd.glom.collect
res7: Array[Array[Int]] = Array(Array(1, 2, 3, 4, 5), Array(6, 7, 8, 9, 10, 11))

scala>
```

Let's visualize the above collect operation in the Spark WebUI(if you are using spark locally, navigate to https://localhost:4040 to see the spark webui or if spark is being accessed via a cluster navigate to your cluster specific localhost webUI)
Note: Spark shell notifies a port number before creating a Sparksession.

Since 2 partitions are present, 2 executors would be launched for this action(Depends if you have that cores or executor), as shown below,



How to see number of Partitions in a Dataframe ?

The above method used for an RDD can also be applied to a dataframe. Let's convert the RDD to a Dataframe and apply the same method. We can see that the same number of partitions are present.

```
# val df = rdd.toDF("number")
# converting rdd to Dataframe
```

```
df.rdd.glom.collect
```

```
scala> df.rdd.glom.collect
res12: Array[Array[org.apache.spark.sql.Row]] = Array(Array([1], [2], [3], [4], [5]), Array([6], [7], [8], [9], [10], [11]))

scala>
```

Now, you must've had the question in your mind as to how spark partitions the data when reading textfiles. Let's read a simple textfile and see the number of partitions here.

```
val peopleDF = spark.read.format("csv").option("sep", ",").option("inferSchema", "true").option("header", "true").load("/software/population.csv")

peopleDF.rdd.getNumPartitions
```

```
scala> val peopleDF = spark.read.format("csv").option("sep", ",").option("inferSchema", "true").option("header", "true").load("/software/popul
ation.csv")
peopleDF: org.apache.spark.sql.DataFrame = [Country Name: string, Country Code: string ... 2 more fields]

scala> peopleDF
res21: org.apache.spark.sql.DataFrame = [Country Name: string, Country Code: string ... 2 more fields]

scala> peopleDF.rdd.get
getCheckpointFile   getClass   getNumPartitions   getStorageLevel

scala> peopleDF.rdd.getNumPartitions
res22: Int = 1

scala>
```

We can see that only 1 partition is created here. Alright let's break this down,

Spark by default creates 1 partition for every 128 MB of the file.
So if you are reading a file of size 1GB, it creates 10 partitions.

So how to control the number of bytes a single partition can hold ?

The file being read here is less than that of the minimum size, hence only 1 partition is created in this case.

The no. of partitions is determined by `spark.sql.files.maxPartitionBytes` parameter, which is set to 128 MB, by default.

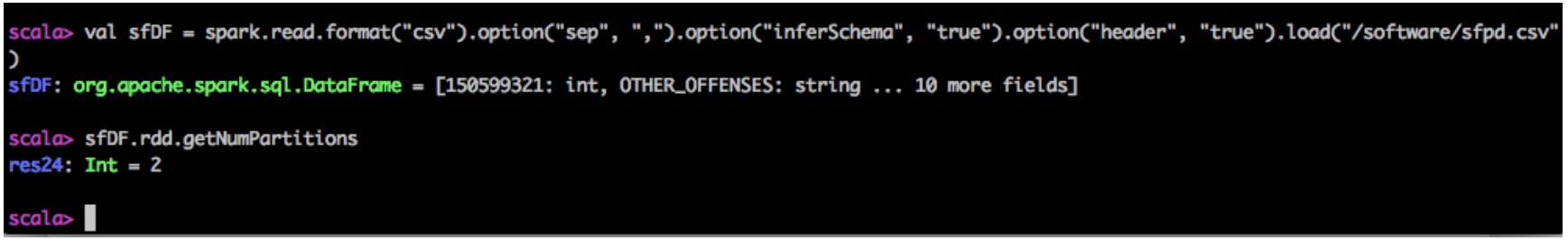
```
spark.conf.get("spark.sql.files.maxPartitionBytes") # -> 128 MB by default
'134217728'
```

Note: *The files being read must be splittable by default for spark to create partitions when reading the file. So, in case of compressed files like snappy, gz or lzo etc, a single partition is created irrespective of the size of the file.*

Let us upload a larger size: 55 mb size

val sfDF = spark.read.format("csv").option("sep", ",").option("inferSchema", "true").option("header", "true").load("/software/sfpd.csv")

sfDF.rdd.getNumPartitions



Let's manually set the `spark.sql.files.maxPartitionBytes` to 5MB, so that we get 12 partitions upon reading the file.

```
spark.conf.set("spark.sql.files.maxPartitionBytes", 5000000)
spark.conf.get("spark.sql.files.maxPartitionBytes")
```

Upload the file again

val sfDF1 = spark.read.format("csv").option("sep", ",").option("inferSchema", "true").option("header", "true").load("/software/sfpd.csv")
sfDF1.rdd.getNumPartitions

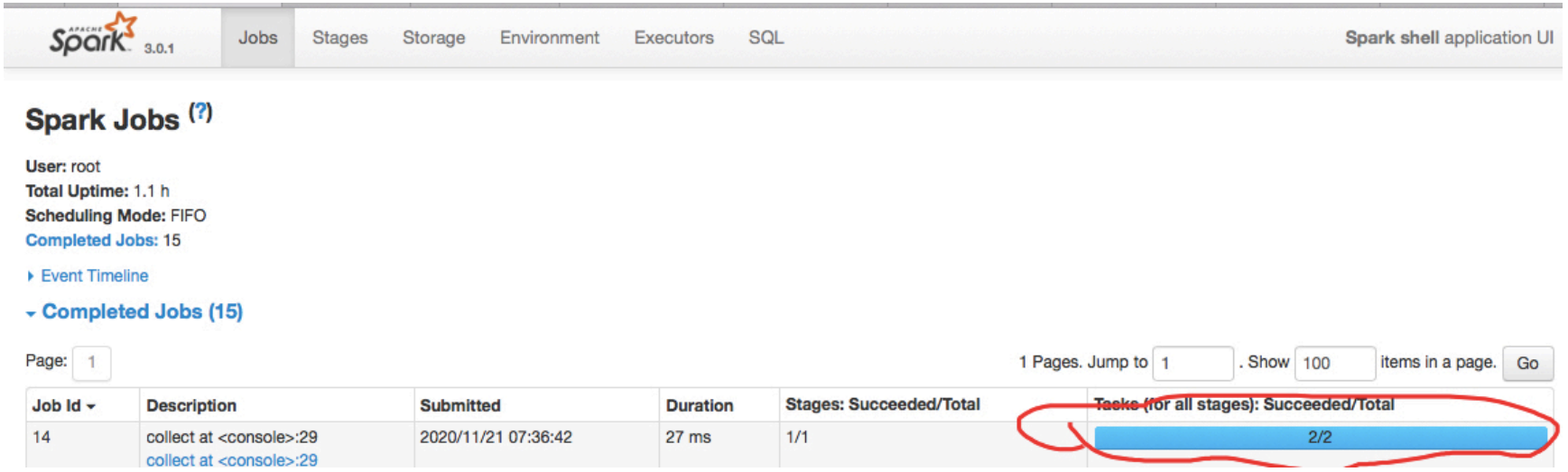
Now, there is 12 Partitions

```
scala> val sfDF1 = spark.read.format("csv").option("sep", ",").option("inferSchema", "true").option("header", "true").load("/software/sfpd.csv
")
sfDF1: org.apache.spark.sql.DataFrame = [150599321: int, OTHER_OFFENSES: string ... 10 more fields]

scala> sfDF1.rdd.getNumPartitions
res27: Int = 12

scala>
```

Let's analyze with a simple example where we create an RDD with a single partition and perform an action on the same.

When the above action is seen on the Spark WebUI, only 2 executors would be issued to process this data.

So, imagine a case where you are processing a huge volume of data without the concept of partitioning, then the entire data would be processed by a single executor taking a lot of time and memory.



Now what if you wish to control these partitions by yourself. This is where the methods of `repartition` and `coalesce` come to effect.

```
df.rdd.getNumPartitions

val partDF = df.repartition(10)

partDF.rdd.getNumPartitions
```

```
scala> df.rdd.getNumPartitions
res39: Int = 2

scala> val partDF = df.repartition
repartition   repartitionByRange

scala> val partDF = df.repartition(10)
partDF: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [number: int]

scala> partDF.rdd.getNumPartitions
res40: Int = 10

scala>
```

After repartitioning , the partition size has increased from 2 to 10 as shown above.

```
spark.conf.get("spark.sql.shuffle.partitions")
```

partDF.rdd.glom.collect

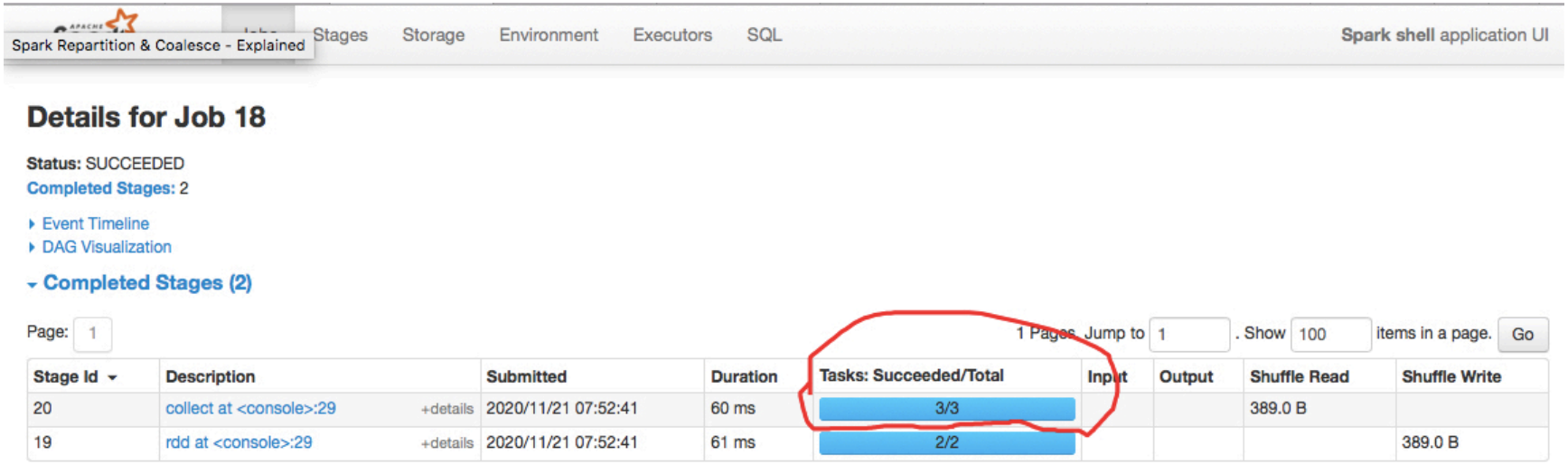
```
scala> partDF.rdd.glom.collect
res42: Array[Array[org.apache.spark.sql.Row]] = Array(Array([8]), Array([3], [9]), Array([5]), Array([2]), Array([4]), Array([1]), Array([11])
, Array([7]), Array([10]), Array([6]))
```

Let's visualise the same on the Spark WebUI

val dfre = df.repartition(3)

dfre.rdd.glom.collect

The above execution when seen on the spark console would show 3 tasks being issued on the collect operation(stage 2).



Unlike repartition, **coalesce doesn't perform a shuffle** to create the partitions.

Let's analyze this using our dataset, let's reduce the number of partitions to 2 now,

partDF.rdd.glom.collect
partDF.rdd.getNumPartitions
Currently its having 10 partitions , let us reduce to 2.

val part3 = partDF.coalesce(2)
part3.rdd.getNumPartitions

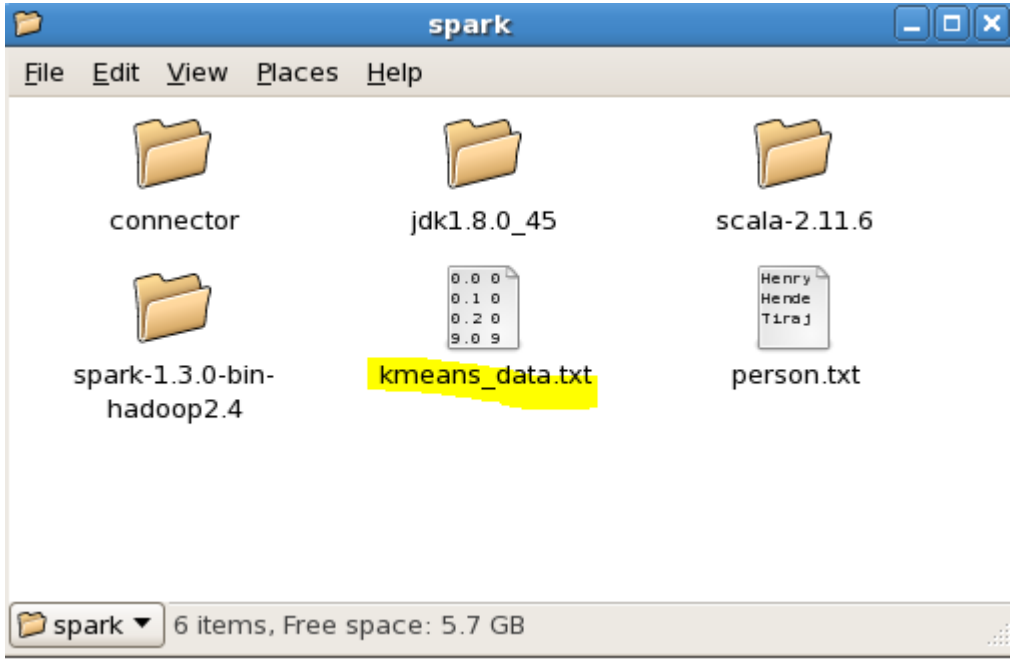
https://datanoon.com/blog/spark_repartition_coalesce/
https://medium.com/@mrpowers/managing-spark-partitions-with-coalesce-and-repartition-4050c57ad5c4
---------- Lab Ends Here ----------

26. Spark -Machine Library - K Clustering – 30 Minutes

Start the spark-shell.

Copy the kmeans_data.txt file to /opt/data folder (All relevant software and resources will be there in the software folder)
It contains the following:

```
0.0 0.0 0.0
0.1 0.1 0.1
0.2 0.2 0.2
9.0 9.0 9.0
9.1 9.1 9.1
9.2 9.2 9.2
```


spark
File Edit View Places Help
connector
jdk1.8.0_45
scala-2.11.6
spark-1.3.0-bin-hadoop2.4
0.0 0
0.1 0
0.2 0
9.0 9
kmeans_data.txt
Henry
Hende
Tiraj
person.txt
spark
6 items, Free space: 5.7 GB

Open one terminal and enter the following command (spark-shell)

```
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>
```

```
// Load and parse the data
val data = sc.textFile("/opt/data/kmeans_data.txt")
```

```
scala> val data = sc.textFile("/spark/kmeans_data.txt")
15/06/07 20:27:44 INFO MemoryStore: ensureFreeSpace(190475) called with curMem=6
60494, maxMem=280248975
15/06/07 20:27:44 INFO MemoryStore: Block broadcast_5 stored as values in memory
 (estimated size 186.0 KB, free 266.5 MB)
15/06/07 20:27:44 INFO MemoryStore: ensureFreeSpace(23141) called with curMem=85
0969, maxMem=280248975
15/06/07 20:27:44 INFO MemoryStore: Block broadcast_5_piece0 stored as bytes in
memory (estimated size 22.6 KB, free 266.4 MB)
15/06/07 20:27:44 INFO BlockManagerInfo: Added broadcast_5_piece0 in memory on h
t.com:52061 (size: 22.6 KB, free: 267.2 MB)
15/06/07 20:27:44 INFO BlockManagerMaster: Updated info of block broadcast_5_pie
ce0
15/06/07 20:27:44 INFO SparkContext: Created broadcast 5 from textFile at <conso
le>:36
data: org.apache.spark.rdd.RDD[String] = /spark/kmeans_data.txt MapPartitionsRDD
[23] at textFile at <console>:36

scala>
```

```
val parsedData = data.map(s => Vectors.dense(s.split(' ').map(_.toDouble))).cache()
```

```
scala> val parsedData = data.map(s => Vectors.dense(s.split(' ').map(_.toDouble)
)).cache()
15/06/07 20:28:17 INFO BlockManager: Removing broadcast 4
15/06/07 20:28:17 INFO BlockManager: Removing block broadcast_4_piece0
15/06/07 20:28:17 INFO MemoryStore: Block broadcast_4_piece0 of size 3463 droppe
d from memory (free 279378328)
15/06/07 20:28:17 INFO BlockManagerInfo: Removed broadcast_4_piece0 on ht.com:36
626 in memory (size: 3.4 KB, free: 267.2 MB)
15/06/07 20:28:17 INFO BlockManagerInfo: Removed broadcast_4_piece0 on ht.com:52
061 in memory (size: 3.4 KB, free: 267.2 MB)
15/06/07 20:28:17 INFO BlockManagerMaster: Updated info of block broadcast_4_pie
ce0
15/06/07 20:28:17 INFO BlockManager: Removing block broadcast_4
15/06/07 20:28:17 INFO MemoryStore: Block broadcast_4 of size 6400 dropped from
memory (free 279384728)
15/06/07 20:28:17 INFO ContextCleaner: Cleaned broadcast 4
parsedData: org.apache.spark.rdd.RDD[org.apache.spark.mllib.linalg.Vector] = Map
PartitionsRDD[24] at map at <console>:38

scala>
```

```
// Cluster the data into two classes using KMeans
val numClusters = 2
val numIterations = 20
val clusters = KMeans.train(parsedData, numClusters, numIterations)
```

```
15/06/07 20:29:19 INFO DAGScheduler: Stage 17 (collectAsMap at KMeans.scala:214)
 finished in 0.039 s
15/06/07 20:29:19 INFO DAGScheduler: Job 15 finished: collectAsMap at KMeans.sca
la:214, took 0.249027 s
15/06/07 20:29:20 INFO KMeans: Run 0 finished in 1 iterations
15/06/07 20:29:20 INFO KMeans: Iterations took 0.414 seconds.
15/06/07 20:29:20 INFO KMeans: KMeans converged in 1 iterations.
15/06/07 20:29:20 INFO KMeans: The cost for the best run is 0.11999999999994547.
15/06/07 20:29:20 INFO MapPartitionsRDD: Removing RDD 25 from persistence list
15/06/07 20:29:20 INFO BlockManager: Removing RDD 25
clusters: org.apache.spark.mllib.clustering.KMeansModel = org.apache.spark.mllib
.clustering.KMeansModel@11cada9

scala>
```

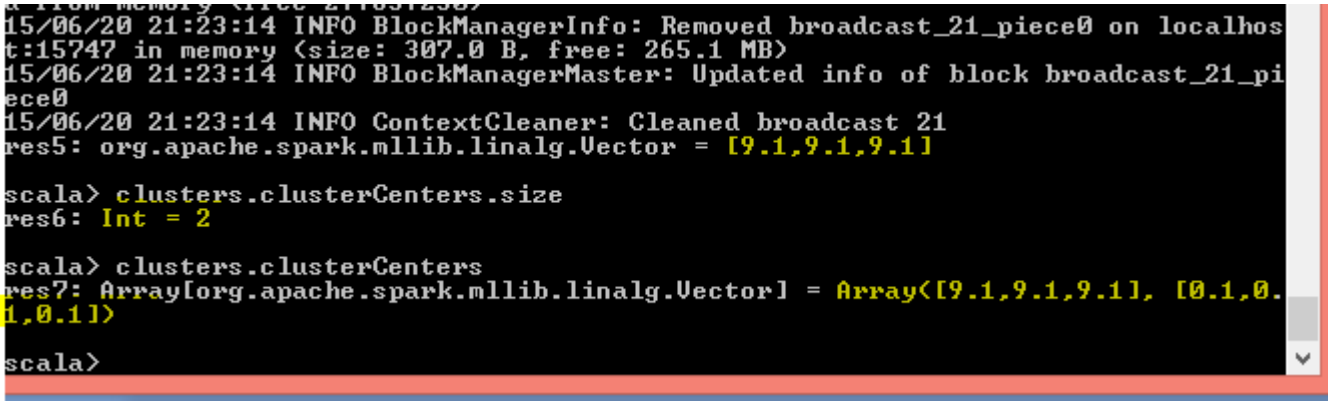
```
// Evaluate clustering by computing Within Set Sum of Squared Errors
val WSSSE = clusters.computeCost(parsedData)
```

```
 all completed, from pool
15/06/07 20:30:07 INFO DAGScheduler: Stage 18 (sum at KMeansModel.scala:56) fini
shed in 0.001 s
15/06/07 20:30:07 INFO DAGScheduler: Job 16 finished: sum at KMeansModel.scala:5
6, took 0.126325 s
WSSSE: Double = 0.11999999999994547
```

```
println("Within Set Sum of Squared Errors = " + WSSSE)
```

```
scala> println("Within Set Sum of Squared Errors = " + WSSSE)
Within Set Sum of Squared Errors = 0.11999999999994547

scala>
```

```
clusters.clusterCenters.head
// You can get the size of the cluster with the following command
clusters.clusterCenters.size
// You can get the center of the two cluster with the following command
clusters.clusterCenters
```



---------- Lab Ends Here ----------

27. Spark MLLib - Predict Store Sales with ML Pipelines

We are going to set up a machine learning pipeline to cover everything from the preprocessing all the way to making and saving predictions. In a sense, this is a "full-stack" machine learning project that's probably fairly similar to something we might do in the real world. Spark's ML Pipelines API is going to make it very easy for us to do this.
Here's a short description about what we're working with:
Rossmann operates over 3,000 drug stores in 7 European countries. Currently, Rossmann store managers are tasked with predicting their daily sales for up to six weeks in advance. Store sales are influenced by many factors, including promotions, competition, school and state holidays, seasonality, and locality. With thousands of individual managers predicting sales based on their unique circumstances, the accuracy of results can be quite varied. In their first Kaggle competition, Rossmann is challenging you to predict 6 weeks of daily sales for 1,115 stores located across Germany. Reliable sales forecasts enable store managers to create effective staff schedules that increase productivity and motivation. By helping Rossmann create a robust prediction model, you will help store managers stay focused on what's most important to them: their customers and their teams!
Review of Linear Regression
There are several assumptions (likely invalidated by our data) that are made by the model.

- That our input data is drawn from a multivariate normal distribution, ie that our variables are independent and normally distributed
- Observations are independent of one another
- A Linear, additive relationship between our input variables and our output variables
- Homoscedasticity of the error terms, or that the error terms should be distributed normally around the regression line

- Our variables are measured without systematic error (and like the point above, that the error values are drawn from some normal process, not caused by a confounding variable)

Let's review those assumptions in this case. We're violating the first one because we our SchoolHoliday and StateHoliday variables are likely to be correlated. We're working with Time Series data so we're likely violating the second one as well. In general, finding problems that strictly fit the linear relationship is difficult and this problem is no exception, so we're likely violating the third assumption as well. The last two we can hope for but again, we're likely violating. With all that being said, we should still at least experiment with this model because it's so well understood and it makes for a strong baseline predictor for future exploration. This aligns with the Concepts of Structural Risk Minimization(if you don't understand SRM, don't worry about it). Now let's define the loss function for linear regression. In linear regression, we hope to minimize the squared error, defined as: L_n = ||Xw^T - y||^2 Where X is our input Matrix, w is our weight vector and y is our output vector. There is a closed form solution to this problem that can be found by inverting the inner product but Spark, because of it's orientation towards big data doesn't actually look to solve the problem this way. Spark leverages gradient descent methods to efficiently descend down the gradient of risk function to arrive at (hopefully) a solution near the global minimum

Create a new Note in Zeppelin Console, PredictStoreML

http://www.sparktutorials.net/Spark+MLLib+-+Predict+Store+Sales+with+ML+Pipelines

28. Installing Jupyter with Spark.

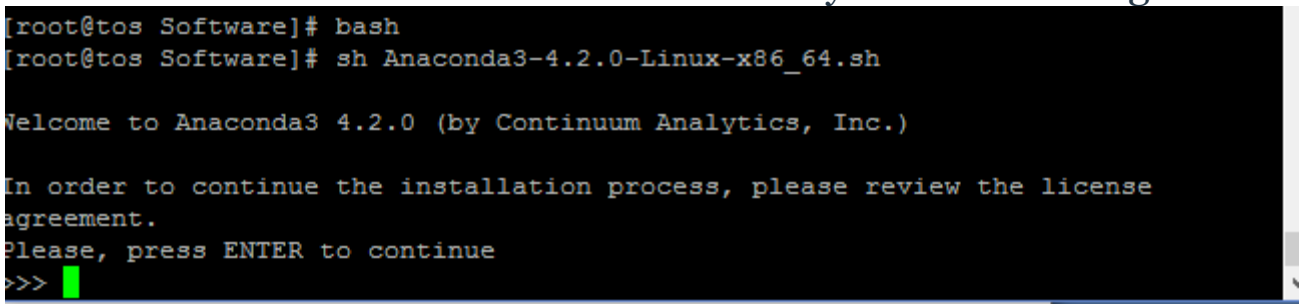
Anaconda 4.2.0
For Linux
Anaconda is BSD licensed which gives you permission to use Anaconda commercially and for redistribution.
Changelog
Download the installer
Optional: Verify data integrity with MD5 or SHA-256 More info
In your terminal window type one of the below and follow the instructions:
Python 3.5 version

```
bash Anaconda3-4.2.0-Linux-x86_64.sh
```

Python 2.7 version

```
bash Anaconda2-4.2.0-Linux-x86_64.sh
```

NOTE: Include the "bash" command even if you are not using the bash shell.



```
Anaconda3 will now be installed into this location:
/root/anaconda3

  - Press ENTER to confirm the location
  - Press CTRL-C to abort the installation
  - Or specify a different location below

[/root/anaconda3] >>> /anaconda3
```

```
Do you wish the installer to prepend the Anaconda3 install location
to PATH in your /root/.bashrc ? [yes|no]
[no] >>> yes

Prepending PATH=/anaconda3/bin to PATH in /root/.bashrc
A backup will be made to: /root/.bashrc-anaconda3.bak


For this change to become active, you have to open a new terminal.

Thank you for installing Anaconda3!

Share your notebooks and packages on Anaconda Cloud!
Sign up for free: https://anaconda.org

[root@tos Software]#
```

#bash
Run the following command to install Jupyter.
#conda install jupyter

```
[root@tos Software]# conda install jupyter
Fetching package metadata .......
Solving package specifications: ..........

Package plan for installation in environment /anaconda3:

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    conda-env-2.6.0            |                0          502 B
    requests-2.12.4            |           py35_0          800 KB
    pyopenssl-16.2.0           |           py35_0           70 KB
    conda-4.3.7                |           py35_0          491 KB
    ------------------------------------------------------------
                                           Total:          1.3 MB
```

```
The following packages will be UPDATED:

    conda:     4.2.9-py35_0  --> 4.3.7-py35_0
    pyopenssl: 16.0.0-py35_0 --> 16.2.0-py35_0
    requests:  2.11.1-py35_0 --> 2.12.4-py35_0

Proceed ([y]/n)? y

Fetching packages ...
conda-env-2.6. 100% |################################| Time: 0:00:00 129.08 kB/s
requests-2.12. 100% |################################| Time: 0:00:11  70.63 kB/s
pyopenssl-16.2 100% |################################| Time: 0:00:01  53.66 kB/s
conda-4.3.7-py 100% |################################| Time: 0:00:04 104.02 kB/s
Extracting packages ...
[      COMPLETE      ]|##################################################| 100%
Unlinking packages ...
[      COMPLETE      ]|##################################################| 100%
Linking packages ...
[      COMPLETE      ]|##################################################| 100%
dbus post-link :: /etc/machine-id not found ..
dbus post-link :: .. using /proc/sys/kernel/random/boot_id
[root@tos Software]#
```

Anaconda 4.2.0

For Windows

Anaconda is BSD licensed which gives you permission to use Anaconda commercially and for redistribution.

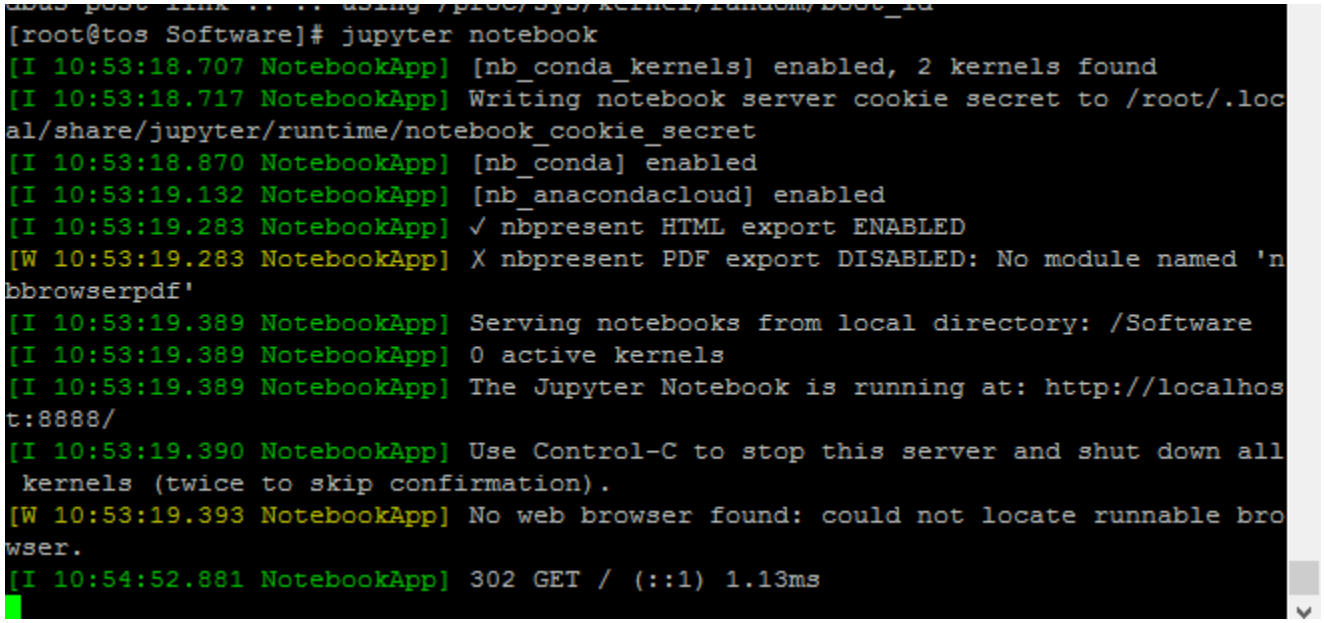
Changelog

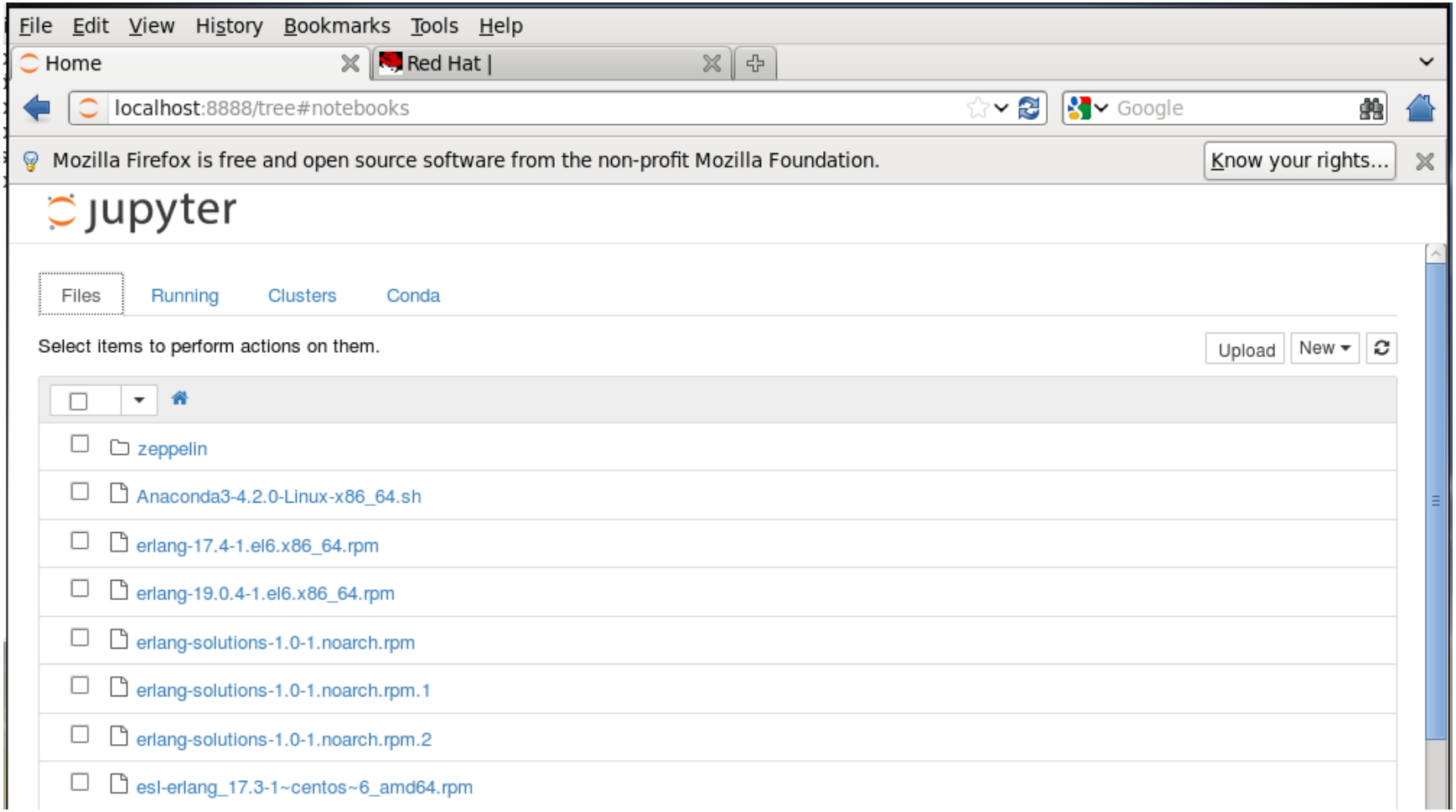
Download the installer

Optional: Verify data integrity with MD5 or SHA-256 More info

Double-click the **.exe** file to install Anaconda and follow the instructions on the screen

Anaconda conveniently installs Python, the Jupyter Notebook, and other commonly used packages for scientific computing and data science.
Congratulations, you have installed Jupyter Notebook. To run the notebook:
jupyter notebook





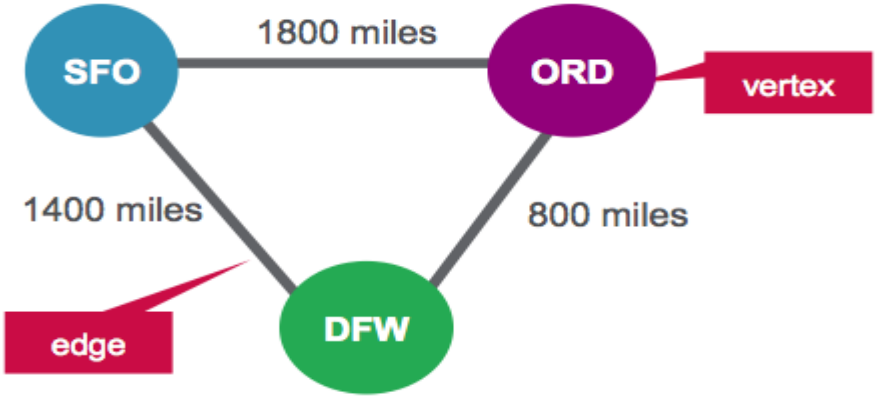
https://community.hortonworks.com/articles/75551/installing-and-exploring-spark-20-with-jupyter-not.html

29. Use GraphX to analyze flight data

As a starting simple example, we will analyze three flights. For each flight, we have the following information:

Originating Airport	Destination Airport	Distance
SFO	ORD	1800 miles
ORD	DFW>	800 miles
DFW	SFO>	1400 miles

In this scenario, we are going to represent the airports as vertices and routes as edges. For our graph we will have three vertices, each representing an airport. The distance between the airports is a route property, as shown below:



VERTEX TABLE FOR AIRPORTS

ID	Property
1	SFO
2	ORD
3	DFW

EDGES TABLE FOR ROUTES

SrcId	DestId	Property
1	2	1800
2	3	800
3	1	1400

Execute Using Zepplin or /spark-shell --master local[2] . You need to execute with root credentials.
Create New Note
DEFINE VERTICES
First we will import the GraphX packages.
import org.apache.spark._

```
import org.apache.spark.rdd.RDD
// import classes required for using GraphX
import org.apache.spark.graphx._
```

We define airports as vertices. Vertices have an Id and can have properties or attributes associated with them. Each vertex consists of:
Vertex id → Id (Long)
Vertex Property → name (String)
VERTEX TABLE FOR AIRPORTS

ID	Property(V)
1	SFO

We define an RDD with the above properties that is then used for the vertexes.

```
// create vertices RDD with ID and Name
val vertices=Array((1L, ("SFO")),(2L, ("ORD")),(3L,("DFW")))
val vRDD= sc.parallelize(vertices)
// [You can view a sample data using take method]
vRDD.take(1)
// Array((1,SFO))

// Defining a default vertex called nowhere
val nowhere = "nowhere"
```

DEFINE EDGES

Edges are the routes between airports. An edge must have a source, a destination, and can have properties. In our example, an edge consists of:

Edge origin id → src (Long)

Edge destination id → dest (Long)

Edge Property distance → distance (Long)

EDGES TABLE FOR ROUTES

srcid	destid	Property(E)
1	12	1800

We define an RDD with the above properties that is then used for the edges. The edge RDD has the form (src id, dest id, distance).

```
// create routes RDD with srcid, destid, distance
val edges = Array(Edge(1L,2L,1800),Edge(2L,3L,800),Edge(3L,1L,1400))
val eRDD= sc.parallelize(edges)

eRDD.take(2)
// Array(Edge(1,2,1800), Edge(2,3,800))
```

CREATE PROPERTY GRAPH

To create a graph, you need to have a Vertex RDD, Edge RDD, and a Default vertex. Create a property graph called graph.

```
// define the graph
val graph = Graph(vRDD,eRDD, nowhere)
// graph vertices
```

```
graph.vertices.collect.foreach(println)
// (2,ORD)
// (1,SFO)
// (3,DFW)

// graph edges
graph.edges.collect.foreach(println)

// Edge(1,2,1800)
// Edge(2,3,800)
// Edge(3,1,1400)
1. How many airports are there?
// How many airports?
val numairports = graph.numVertices
// Long = 3
2. How many routes are there?
// How many routes?
val numroutes = graph.numEdges
// Long = 3
3. which routes > 1000 miles distance?
// routes > 1000 miles distance?
graph.edges.filter { case Edge(src, dst, prop) => prop > 1000 }.collect.foreach(println)
// Edge(1,2,1800)
// Edge(3,1,1400)
```

4. The EdgeTriplet class extends the Edge class by adding the srcAttr and dstAttr members which contain the source and destination properties, respectively.

```
// triplets
graph.triplets.take(3).foreach(println)
((1,SFO),(2,ORD),1800)
((2,ORD),(3,DFW),800)
((3,DFW),(1,SFO),1400)
```

5. Sort and print out the longest distance routes

```
// print out longest routes
graph.triplets.sortBy(_.attr, ascending=false).map(triplet =>
    "Distance " + triplet.attr.toString + " from " + triplet.srcAttr + " to " + triplet.dstAttr +
".").collect.foreach(println)
```

The complete Program:
Spark-Shell Windows:

```
Spark context Web UI available at http://10.10.20.20:4040
Spark context available as 'sc' (master = local[2], app id = local-1522572991959
).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.1.0-mapr-1707
      /_/

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_45)
Type in expressions to have them evaluated.
Type :help for more information.

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

scala> import org.apache.spark.rdd.RDD
import org.apache.spark.rdd.RDD

scala> // import classes required for using GraphX
```

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

scala> val vertices=Array((1L, ("SFO")),(2L, ("ORD")),(3L,("DFW")))
vertices: Array[(Long, String)] = Array((1,SFO), (2,ORD), (3,DFW))

scala> val vRDD= sc.parallelize(vertices)
vRDD: org.apache.spark.rdd.RDD[(Long, String)] = ParallelCollectionRDD[0] at par
allelize at <console>:33

scala> vRDD.take(1)
res0: Array[(Long, String)] = Array((1,SFO))

scala> val nowhere = "nowhere"
nowhere: String = nowhere

scala> val edges = Array(Edge(1L,2L,1800),Edge(2L,3L,800),Edge(3L,1L,1400))
edges: Array[org.apache.spark.graphx.Edge[Int]] = Array(Edge(1,2,1800), Edge(2,3
,800), Edge(3,1,1400))

scala> val eRDD= sc.parallelize(edges)
eRDD: org.apache.spark.rdd.RDD[org.apache.spark.graphx.Edge[Int]] = ParallelColl
ectionRDD[1] at parallelize at <console>:33
```

```
scala> vRDD.take(1)
res0: Array[(Long, String)] = Array((1,SFO))

scala> val nowhere = "nowhere"
nowhere: String = nowhere

scala> val edges = Array(Edge(1L,2L,1800),Edge(2L,3L,800),Edge(3L,1L,1400))
edges: Array[org.apache.spark.graphx.Edge[Int]] = Array(Edge(1,2,1800), Edge(2,3
,800), Edge(3,1,1400))

scala> val eRDD= sc.parallelize(edges)
eRDD: org.apache.spark.rdd.RDD[org.apache.spark.graphx.Edge[Int]] = ParallelColl
ectionRDD[1] at parallelize at <console>:33

scala>

scala> eRDD.take(2)
res1: Array[org.apache.spark.graphx.Edge[Int]] = Array(Edge(1,2,1800), Edge(2,3,
800))

scala> val graph = Graph(vRDD,eRDD, nowhere)
graph: org.apache.spark.graphx.Graph[String,Int] = org.apache.spark.graphx.impl.
GraphImpl@5b265379
```

```
scala> graph.vertices.collect.foreach(println)
18/04/01 14:30:07 WARN Executor: 1 block locks were not released by TID = 8:
[rdd_9_1]
18/04/01 14:30:07 WARN Executor: 1 block locks were not released by TID = 7:
[rdd_9_0]
(2,ORD)
(1,SFO)
(3,DFW)

scala> graph.edges.collect.foreach(println)
18/04/01 14:30:19 WARN Executor: 1 block locks were not released by TID = 9:
[rdd_12_0]
18/04/01 14:30:19 WARN Executor: 1 block locks were not released by TID = 10:
[rdd_12_1]
Edge(1,2,1800)
Edge(2,3,800)
Edge(3,1,1400)

scala>

scala> val numairports = graph.numVertices
numairports: Long = 3
```

```
scala> val numroutes = graph.numEdges
numroutes: Long = 3

scala> graph.edges.filter { case Edge(src, dst, prop) => prop > 1000 }.collect.f
oreach(println)
18/04/01 14:30:48 WARN Executor: 1 block locks were not released by TID = 15:
[rdd_12_0]
18/04/01 14:30:48 WARN Executor: 1 block locks were not released by TID = 16:
[rdd_12_1]
Edge(1,2,1800)
Edge(3,1,1400)

scala> graph.triplets.take(3).foreach(println)
18/04/01 14:30:56 WARN Executor: 1 block locks were not released by TID = 20:
[rdd_12_1]
((1,SFO),(2,ORD),1800)
((2,ORD),(3,DFW),800)
((3,DFW),(1,SFO),1400)

scala> graph.triplets.sortBy(_.attr, ascending=false).map(triplet =>
     |       "Distance " + triplet.attr.toString + " from " + triplet.srcAttr + "
 to " + triplet.dstAttr + ".").collect.foreach(println)
Distance 1800 from SFO to ORD.
Distance 1400 from DFW to SFO.
Distance 800 from ORD to DFW.

scala>
```

GraphX

```
import org.apache.spark._
import org.apache.spark.rdd.RDD
// import classes required for using GraphX
import org.apache.spark.graphx._
```

```
import org.apache.spark._
import org.apache.spark.rdd.RDD
import org.apache.spark.graphx._
```

Took 3 sec. Last updated by anonymous at January 23 2017, 11:52:35 PM.

```
// create vertices RDD with ID and Name
val vertices=Array((1L, ("SFO")),(2L, ("ORD")),(3L,("DFW")))
val vRDD= sc.parallelize(vertices)
vRDD.take(1)
// Array((1,SFO))

// Defining a default vertex called nowhere
val nowhere = "nowhere"
```

```
vertices: Array[(Long, String)] = Array((1,SFO), (2,ORD), (3,DFW))
vRDD: org.apache.spark.rdd.RDD[(Long, String)] = ParallelCollectionRDD[0] at parallelize at <console>:43
res5: Array[(Long, String)] = Array((1,SFO))
nowhere: String = nowhere
```

Took 11 sec. Last updated by anonymous at January 23 2017, 11:53:14 PM.

```
// create routes RDD with srcid, destid, distance
val edges = Array(Edge(1L,2L,1800),Edge(2L,3L,800),Edge(3L,1L,1400))
val eRDD= sc.parallelize(edges)

eRDD.take(2)
// Array(Edge(1,2,1800), Edge(2,3,800))
```

```
edges: Array[org.apache.spark.graphx.Edge[Int]] = Array(Edge(1,2,1800), Edge(2,3,800), Edge(3,1,1400))
eRDD: org.apache.spark.rdd.RDD[org.apache.spark.graphx.Edge[Int]] = ParallelCollectionRDD[1] at parallelize at <console>:43
res8: Array[org.apache.spark.graphx.Edge[Int]] = Array(Edge(1,2,1800), Edge(2,3,800))
```

Took 4 sec. Last updated by anonymous at January 23 2017, 11:53:36 PM.

```
// define the graph
val graph = Graph(vRDD,eRDD, nowhere)
// graph vertices
graph.vertices.collect.foreach(println)
// (2,ORD)
// (1,SFO)
// (3,DFW)

// graph edges
graph.edges.collect.foreach(println)

// Edge(1,2,1800)
// Edge(2,3,800)
// Edge(3,1,1400)
```

```
graph: org.apache.spark.graphx.Graph[String,Int] = org.apache.spark.graphx.impl.GraphImpl@3c4394bb
(1,SFO)
(2,ORD)
(3,DFW)
Edge(1,2,1800)
Edge(2,3,800)
Edge(3,1,1400)
```

Took 5 sec. Last updated by anonymous at January 23 2017, 11:54:23 PM.

```
// How many airports?
val numairports = graph.numVertices
```

```
numairports: Long = 3
```

Took 1 sec. Last updated by anonymous at January 23 2017, 11:54:42 PM.

```
val numroutes = graph.numEdges
```

```
numroutes: Long = 3
```

Took 1 sec. Last updated by anonymous at January 23 2017, 11:54:51 PM.

```
// routes > 1000 miles distance?
graph.edges.filter { case Edge(src, dst, prop) => prop > 1000 }.collect.foreach(println)
```

```
Edge(1,2,1800)
Edge(3,1,1400)
```

Took 3 sec. Last updated by anonymous at January 23 2017, 11:55:03 PM.

```
// triplets
graph.triplets.take(3).foreach(println)
```

```
((1,SFO),(2,ORD),1800)
((2,ORD),(3,DFW),800)
((3,DFW),(1,SFO),1400)
```

Took 2 sec. Last updated by anonymous at January 23 2017, 11:55:36 PM.

```
// print out longest routes
graph.triplets.sortBy(_.attr, ascending=false).map(triplet =>
    "Distance " + triplet.attr.toString + " from " + triplet.srcAttr + " to " + triplet.dstAttr + ".").collect.foreach(println)
```

```
Distance 1800 from SFO to ORD.
Distance 1400 from DFW to SFO.
Distance 800 from ORD to DFW.
```

Took 3 sec. Last updated by anonymous at January 23 2017, 11:55:50 PM.

30. Troubleshooting & Tuning– 60 Minutes

Here, you will learn the following:

- Specify job configuration parameters
- Specify the number of partitions
- Verify job details using Spark web UI

The number of cores can be specified with any of the following:

- `--executor-cores` flag when invoking spark-submit, spark-shell, and pyspark
- `spark.executor.cores` property in the `spark-defaults.conf` file
- on a `SparkConf` object.

Similarly, the heap size can be controlled with any of the following parameters:

- `--executor-memory` flag
- `spark.executor.memory` property.

The `cores` property controls the number of concurrent tasks an executor can run. `--executor-cores 5` means that each executor can run a maximum of five tasks at the same time.

The `--num-executors` command-line flag or `spark.executor.instances` configuration property control the number of executors requested.

Best practice is to avoid setting this property by turning on dynamic allocation with the `spark.dynamicAllocation.enabled` property

The relevant YARN properties are:

`yarn.nodemanager.resource.memory-mb` controls the maximum sum of memory used by the containers on each node.

`yarn.nodemanager.resource.cpu-vcores` controls the maximum sum of cores used by the containers on each node.

```
  </property>
▼<property>
    <name>yarn.nodemanager.resource.memory-mb</name>
    <value>4096</value>
    <final>false</final>
    <source>yarn-site.xml</source>
  </property>
```

```
▼<property>
    <name>yarn.nodemanager.resource.cpu-vcores</name>
    <value>-1</value>
    <final>false</final>
    <source>yarn-default.xml</source>
  </property>
```

YARN may round the requested memory up a little. YARN's `yarn.scheduler.minimum-allocation-mb` and `yarn.scheduler.increment-allocation-mb` properties control the minimum and increment request values respectively.

```
▼<property>
   <name>yarn.scheduler.minimum-allocation-mb</name>
   <value>512</value>
   <final>false</final>
   <source>yarn-site.xml</source>
 </property>
▼<property>
```

```
▼<property>
   <name>yarn.scheduler.increment-allocation-mb</name>
   <value>256</value>
   <final>false</final>
   <source>yarn-site.xml</source>
 </property>
```

Verify the above setting and note down the value for your cluster.

Let us submit the job specifying the configuration.

You need to upload the input file for executing this program to the HDFS cluster.
#hdfs dfs -mkdir /tuneip
#hdfs dfs -copyFromLocal flights20170102.json /tuneip

Open a new sparkshell from a terminal
spark-shell
#export HADOOP_CONF_DIR=/opt/hadoop/etc/hadoop

Only Standalone Mode:
spark-shell --executor-core 2 --executor-memory 250 --num-executors 2

On YARN Mode
spark-shell --master yarn --executor-memory 512mb --executor-cores 2 --num-executors 2

```
import org.apache.spark.sql.types._
import org.apache.spark.sql._
import org.apache.spark.sql.functions._

var file = "hdfs://master0/tuneip/flights20170102.json"

case class Flight(_id: String, dofW: Long, carrier: String, origin: String, dest: String, crsdephour: Long, crsdeptime: Double, depdelay: Double,crsarrtime: Double, arrdelay: Double, crselapsedtime: Double, dist: Double) extends Serializable

val df = spark.read.format("json").option("inferSchema", "true").load(file).as[Flight]

val df2 = df.filter($"depdelay" > 40)
```

df2.take(1)

df2.explain(true)

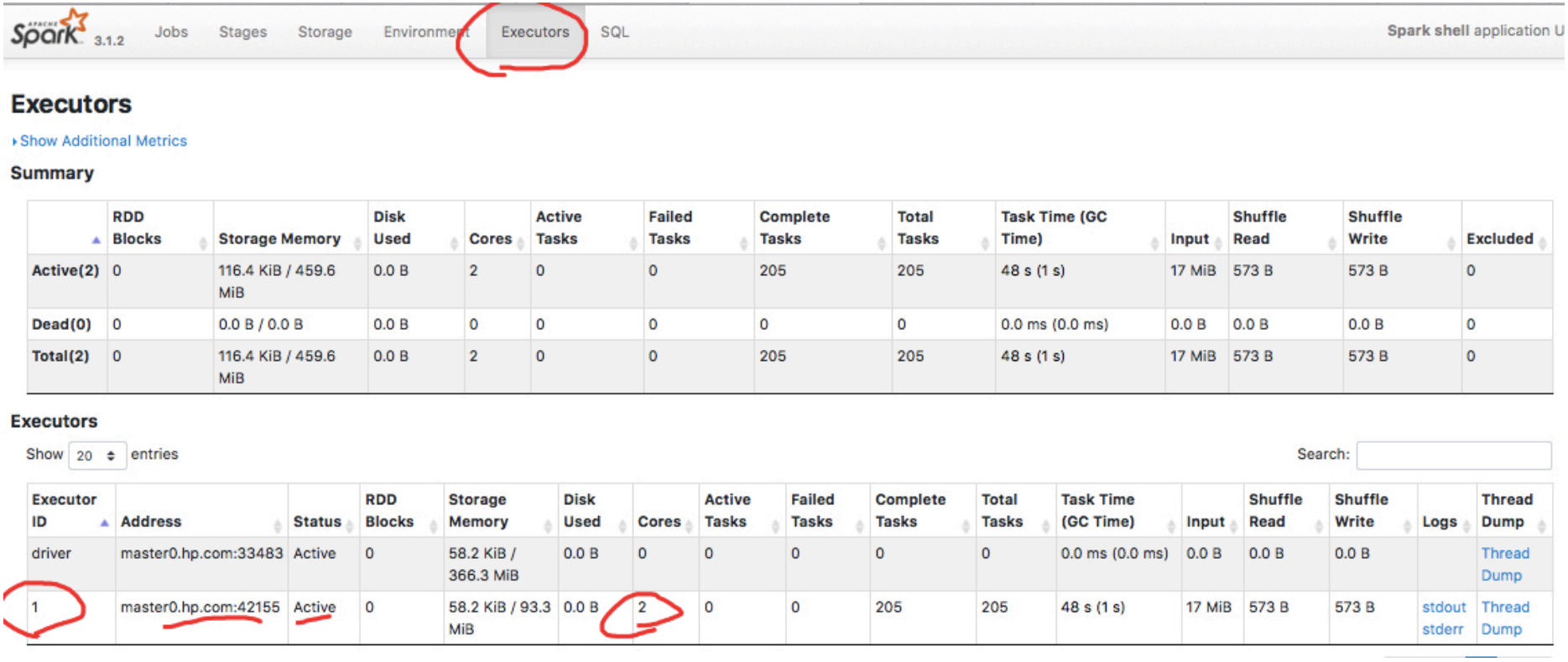
```
scala> var file = "hdfs://master0/tuneip/flights20170102.json"
file: String = hdfs://master0/tuneip/flights20170102.json

scala> val df = spark.read.format("json").option("inferSchema", "true").load(file).as[Flight]
df: org.apache.spark.sql.Dataset[Flight] = [_id: string, arrdelay: double ... 10 more fields]

scala> val df2 = df.filter($"depdelay" > 40)
df2: org.apache.spark.sql.Dataset[Flight] = [_id: string, arrdelay: double ... 10 more fields]

scala> df2.take(1)
res0: Array[Flight] = Array(Flight(ATL_BOS_2017-01-02_16_DL_1210,1,DL,ATL,BOS,16,1616.0,68.0,1852.0,49.0,156.0,946.0))
```

```
scala> df2.explain(true)
== Parsed Logical Plan ==
'Filter ('depdelay > 40)
+- Relation[_id#7,arrdelay#8,carrier#9,crsarrtime#10,crsdephour#11L,crsdeptime#12,crselapsedtime#13,depdelay#14,dest#15,dist#16,dofW#17L,origin#18]
json

== Analyzed Logical Plan ==
_id: string, arrdelay: double, carrier: string, crsarrtime: double, crsdephour: bigint, crsdeptime: double, crselapsedtime: double, depdelay: double, dest: string,
dist: double, dofW: bigint, origin: string
Filter (depdelay#14 > cast(40 as double))
+- Relation[_id#7,arrdelay#8,carrier#9,crsarrtime#10,crsdephour#11L,crsdeptime#12,crselapsedtime#13,depdelay#14,dest#15,dist#16,dofW#17L,origin#18]
json

== Optimized Logical Plan ==
Filter (isnotnull(depdelay#14) AND (depdelay#14 > 40.0))
+- Relation[_id#7,arrdelay#8,carrier#9,crsarrtime#10,crsdephour#11L,crsdeptime#12,crselapsedtime#13,depdelay#14,dest#15,dist#16,dofW#17L,origin#18]
json

== Physical Plan ==
*(1) Filter (isnotnull(depdelay#14) AND (depdelay#14 > 40.0))
+- FileScan json
[_id#7,arrdelay#8,carrier#9,crsarrtime#10,crsdephour#11L,crsdeptime#12,crselapsedtime#13,depdelay#14,dest#15,dist#16,dofW#17L,origin#18] Batched:
false, DataFilters: [isnotnull(depdelay#14), (depdelay#14 > 40.0)], Format: JSON, Location: InMemoryFileIndex[hdfs://master0/tuneip/flights20170102.json],
PartitionFilters: [], PushedFilters: [IsNotNull(depdelay), GreaterThan(depdelay,40.0)], ReadSchema:
struct<_id:string,arrdelay:double,carrier:string,crsarrtime:double,crsdephour:bigint,crsdeptime:d...
```

```
val df3 = df2.groupBy("carrier").count

df3.collect

df3.explain
```

```
scala> val df3 = df2.groupBy("carrier").count
df3: org.apache.spark.sql.DataFrame = [carrier: string, count: bigint]

scala> df3.collect
res2: Array[org.apache.spark.sql.Row] = Array([UA,2420], [AA,757], [DL,1043], [WN,244])

scala>

scala> df3.explain
== Physical Plan ==
*(2) HashAggregate(keys=[carrier#9], functions=[count(1)])
+- Exchange hashpartitioning(carrier#9, 200), ENSURE_REQUIREMENTS, [id=#49]
   +- *(1) HashAggregate(keys=[carrier#9], functions=[partial_count(1)])
      +- *(1) Project [carrier#9]
         +- *(1) Filter (isnotnull(depdelay#14) AND (depdelay#14 > 40.0))
            +- FileScan json [carrier#9,depdelay#14] Batched: false, DataFilters: [isnotnull(depdelay#14), (depdelay#14 > 40.0)],
Format: JSON, Location: InMemoryFileIndex[hdfs://master0/tuneip/flights20170102.json], PartitionFilters: [], PushedFilters: [IsNot
Null(depdelay), GreaterThan(depdelay,40.0)], ReadSchema: struct<carrier:string,depdelay:double>



scala>
```

As observe, it has created 200 partitions.
How many executors? And cores



1 Executor and 2 Cores.

Let us reduce the no of partitions

spark.conf.set("spark.sql.shuffle.partitions",2)
df2.groupBy("carrier").count.collect

```
scala> spark.conf.set("spark.sql.shuffle.partitions",2)

scala> df2.groupBy("carrier").count.collect
res19: Array[org.apache.spark.sql.Row] = Array([WN,244], [DL,1043], [UA,2420], [AA,757])

scala>
```

Spark Jobs (?)

User: root
Total Uptime: 32 min
Scheduling Mode: FIFO
Completed Jobs: 6

▸ Event Timeline

▾ **Completed Jobs (6)**

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
5	collect at <console>:35 collect at <console>:35	2022/06/18 08:43:46	1 s	2/2	4/4
4	collect at <console>:37 collect at <console>:37	2022/06/18 08:38:04	12 s	3/3	205/205

As shown above, partition has been reduce, hence the number of tasks come down and the duration also decreases.

---------- Lab Ends Here ----------

31. Annexure:

Kafka Installation:

Perform only on the first Node or Sparko container.

Steps to be performed:

- Install Kafka and start it along with producer.

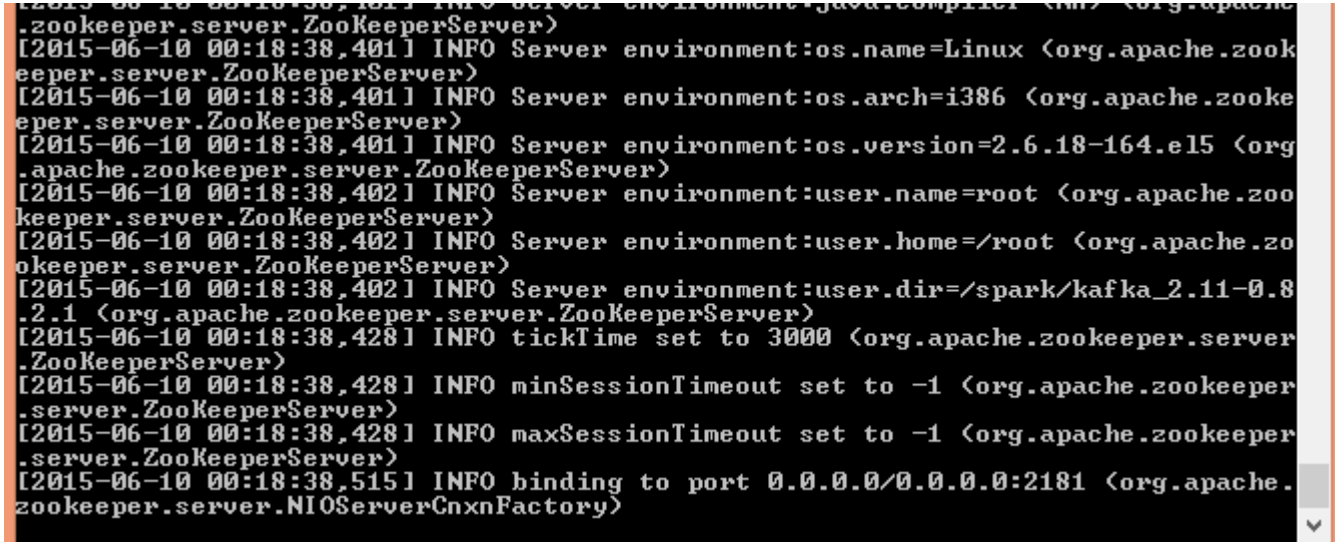
Download Kafka from - https://kafka.apache.org/downloads

Install kafka: kafka_2.13-2.7.0.tgz

```
#tar -xvf kafka_*.tgz  -C /opt
# mv kafka* kafka
#cd /opt/kafka
```

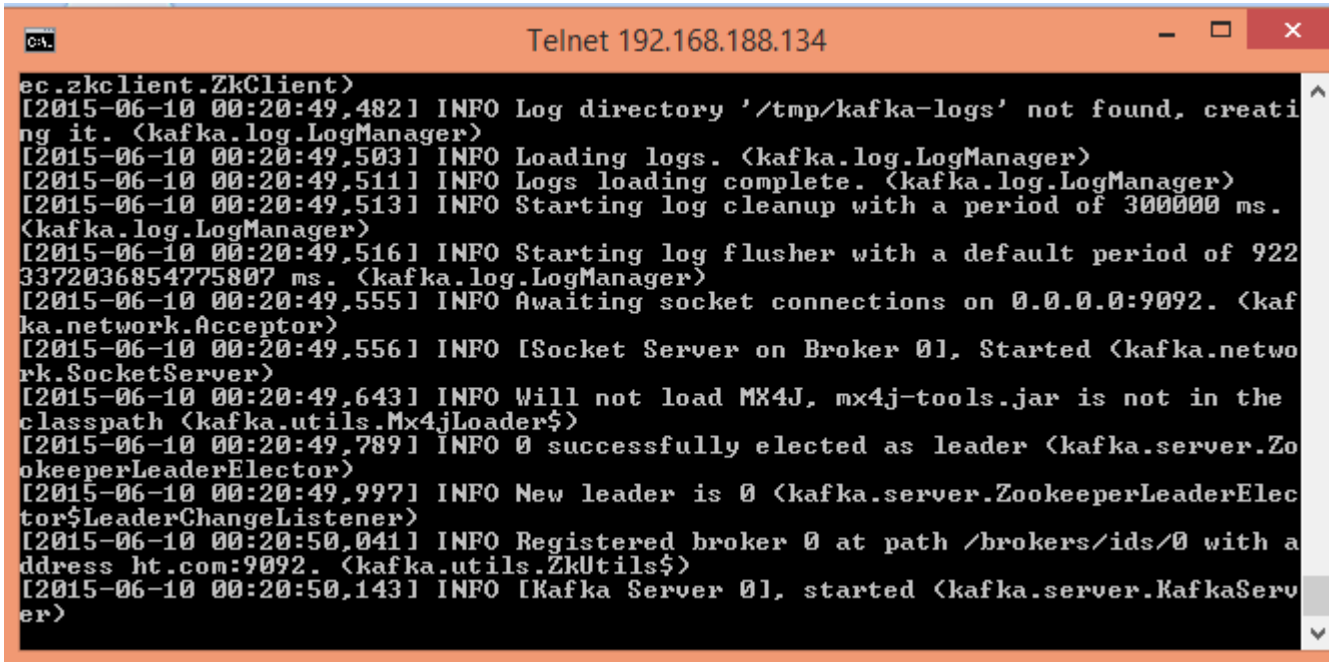
To Start the server, start zookeeper first from a terminal

```
bin/zookeeper-server-start.sh config/zookeeper.properties
```



Now start the Kafka server on other terminal

bin/kafka-server-start.sh config/server.properties



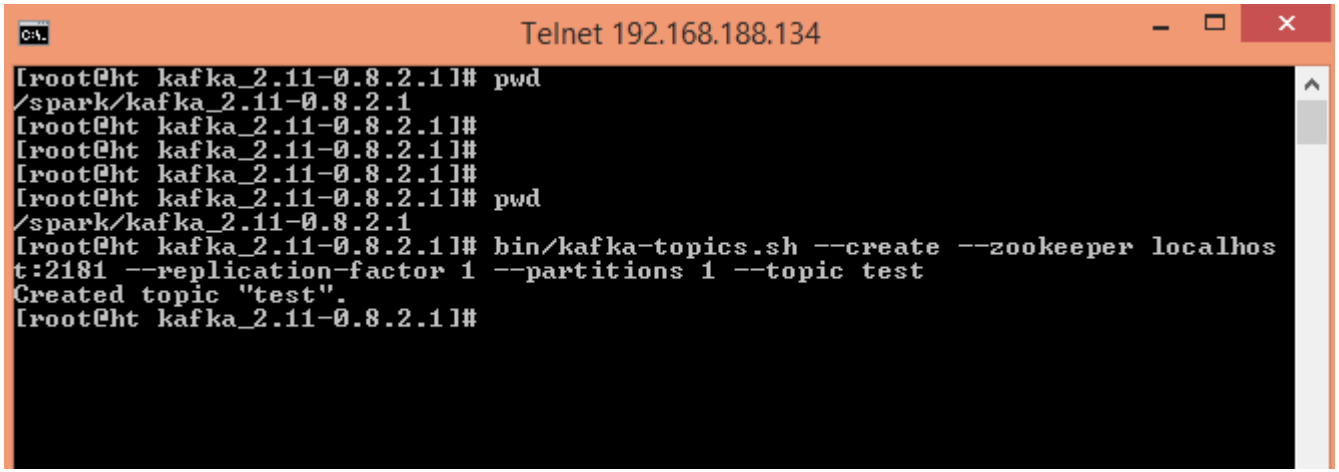
Keep both the console open.

Create a topic

Let's create a topic named "test" with a single partition and only one replica: Open another terminal

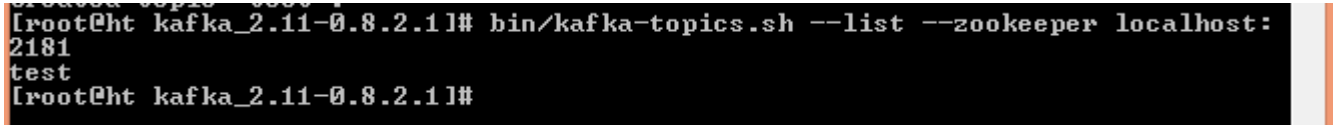
cd /opt/kafka

#bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test



We can now see that topic if we run the list topic command:

> bin/kafka-topics.sh --list --zookeeper localhost:2181



Send some messages

// Open a new console, change directory to the kafka installation folder

```
test
[root@ht kafka_2.11-0.8.2.1]# bin/kafka-console-producer.sh --broker-list local
host:9092 --topic test
[2015-06-10 00:25:31,582] WARN Property topic is not valid (kafka.utils.Verifiab
leProperties)
I am sending a message
```

#bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test

// type some message

Start a consumer

// Open a new console, change directory to the kafka installation folder

#bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test --from-beginning

```
[root@303b68143644 kafka]# bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test --from-
beginning
Hello How r u?
```