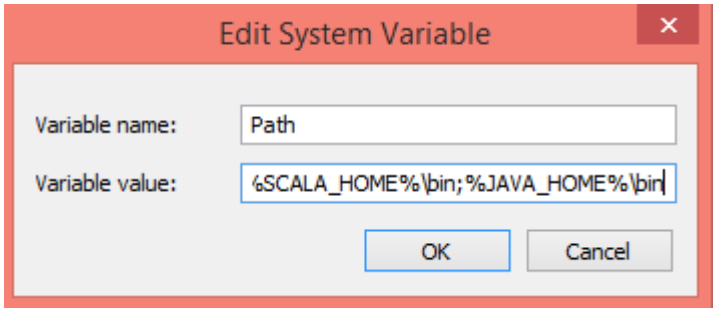
Table of Contents

Last Updated: 19 Mar 2022 (Yellow – Updated)
spark-3.1.2-bin-hadoop3.2.tgz
Hadoop 3.1.2 (hadoop-3.2.2.tar.gz)
zeppelin-0.9.0-bin-all.tgz

- Use the version specified above.
- All Job Monitoring to be done in Spark cluster since it required lower memory
- Zeppelein 0.90 is not compatible with spark 3.1.2.
- Use zeppelin for spark shell examples only.

1. Exercise 1: Spark Installation: Windows:

You need to install java on your system PATH, or the JAVA_HOME environment variable pointing to a Java installation.

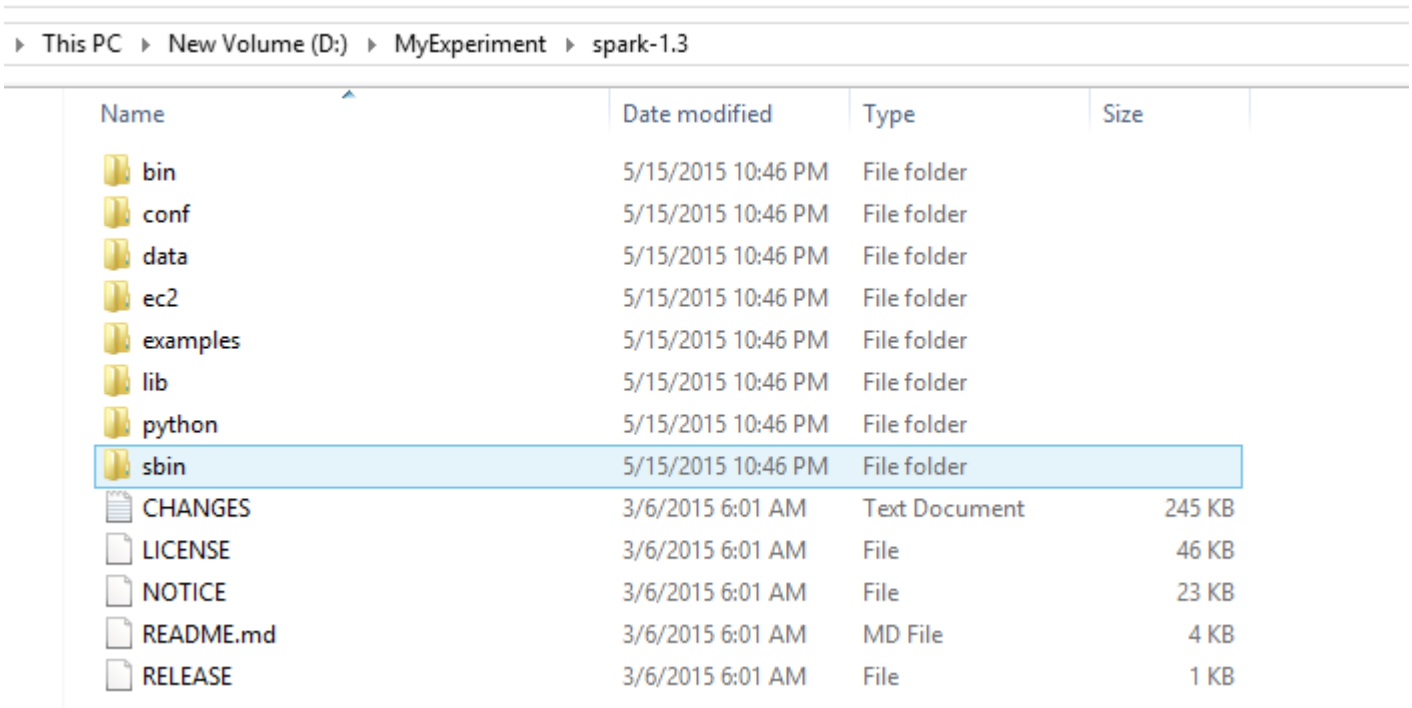



```
C:\Users\henry>echo %JAVA_HOME%
D:\jdk1.8.0_45
```

```
C:\Users\henry>java -version
java version "1.8.0_45"
Java(TM) SE Runtime Environment (build 1.8.0_45-b15)
Java HotSpot(TM) 64-Bit Server VM (build 25.45-b02, mixed mode)
```

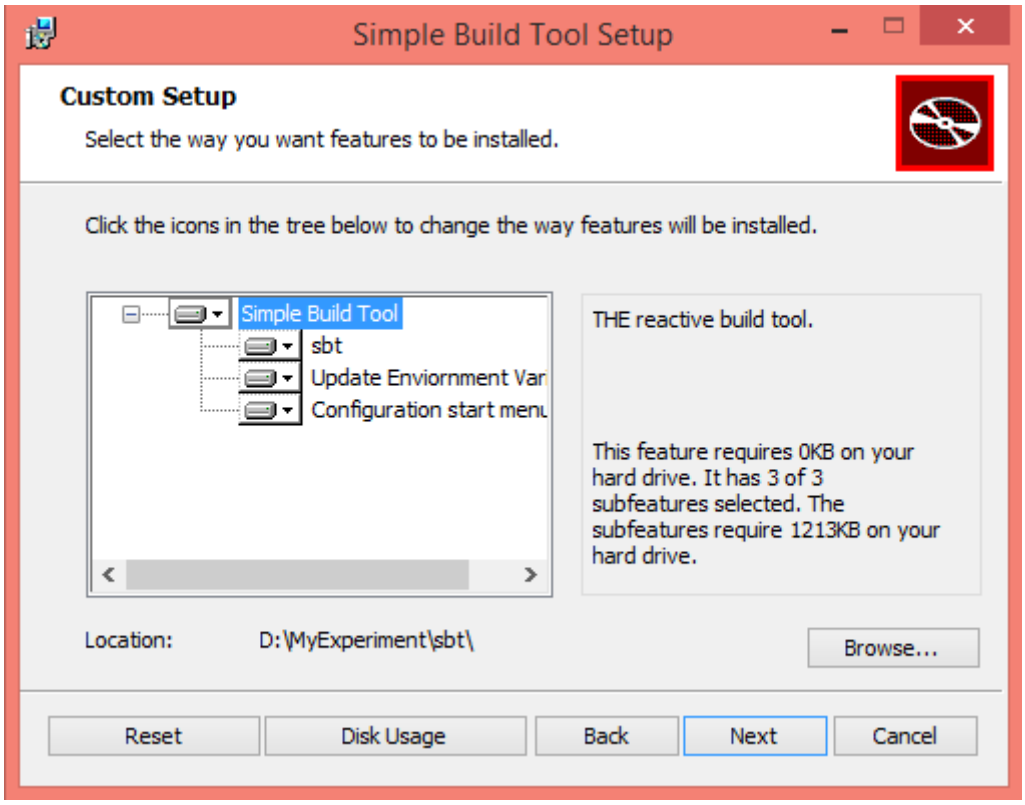
Spark runs on Java 6+ and Python 2.6+.

Unzip spark-1.3.0-bin-hadoop2.4.tar to D:\MyExperiment

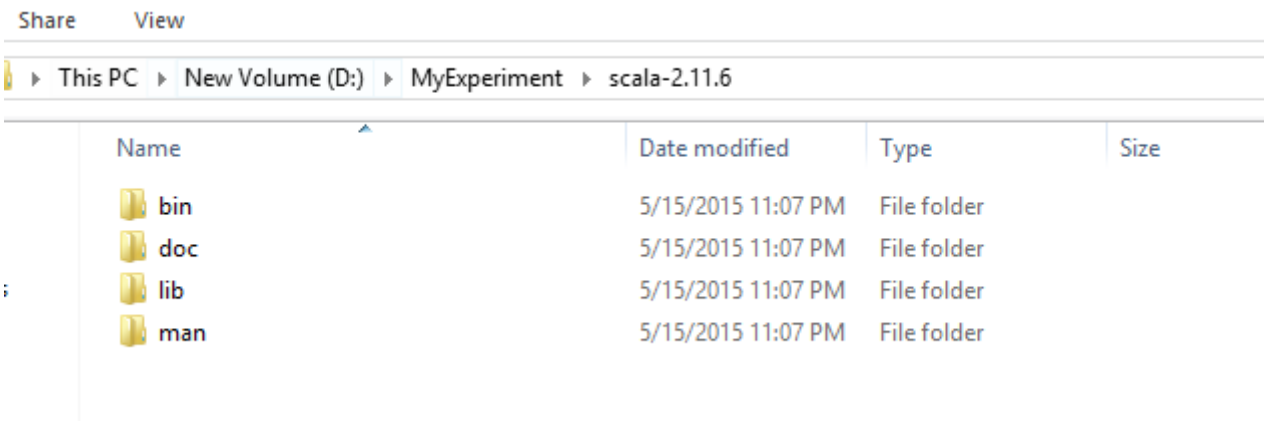


Install sbt.MSI & execute it.

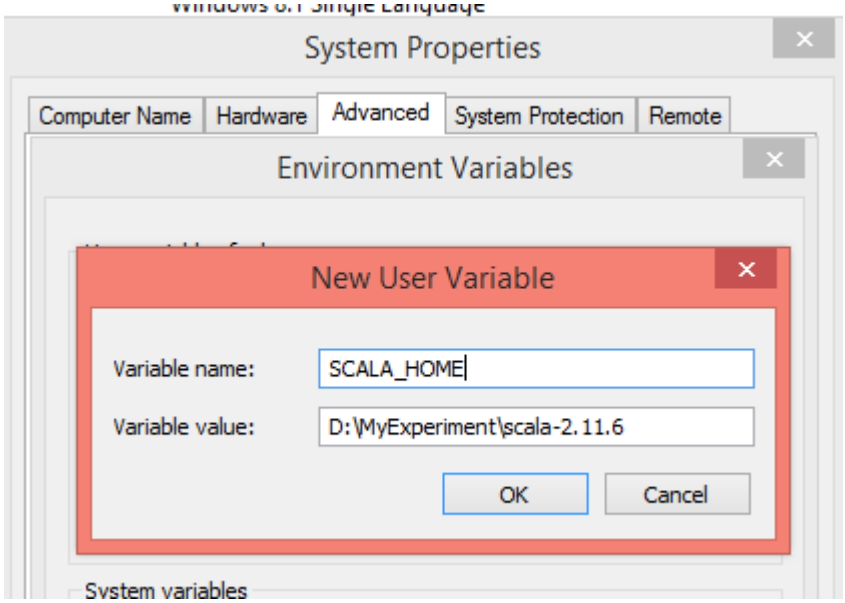
sbt-0.13.8.msi

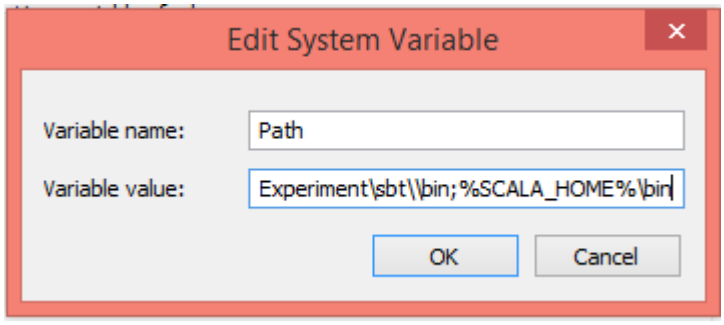


Untar scala-2.11.6



Set SCALA_HOME environment variable & set the PATH variable to the bin directory of scala.





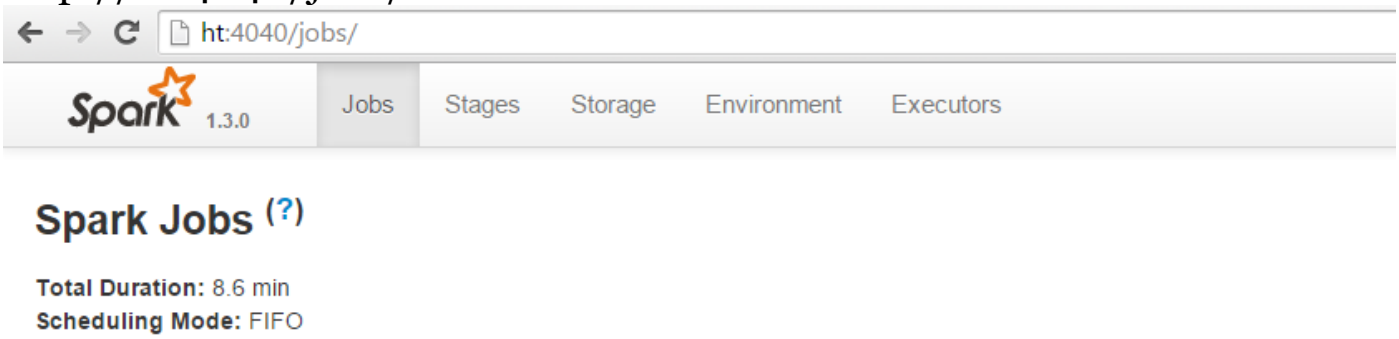
change to the installation directory : D:\MyExperiment\spark-1.3>cd bin and execute
spark-shell

```
D:\MyExperiment\spark-1.3\bin>spark-shell
log4j:WARN No appenders could be found for logger (org.apache.hadoop.metrics2.li
b.MutableMetricsFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more in
fo.
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties

15/05/15 23:22:19 INFO Executor: Using REPL class URI: http://192.168.188.1:4967
2
15/05/15 23:22:19 INFO AkkaUtils: Connecting to HeartbeatReceiver: akka.tcp://sp
arkDriver@ht:49686/user/HeartbeatReceiver
15/05/15 23:22:19 INFO NettyBlockTransferService: Server created on 49706
15/05/15 23:22:19 INFO BlockManagerMaster: Trying to register BlockManager
15/05/15 23:22:19 INFO BlockManagerMasterActor: Registering block manager localh
ost:49706 with 265.1 MB RAM, BlockManagerId(<driver>, localhost, 49706)
15/05/15 23:22:19 INFO BlockManagerMaster: Registered BlockManager
15/05/15 23:22:19 INFO SparkILoop: Created spark context..
Spark context available as sc.
15/05/15 23:22:20 INFO SparkILoop: Created sql context (with Hive support)..
SQL context available as sqlContext.

scala>
```

Create a data set of 1...10000 integers
val data = 1 to 10000

```
scala> val data = 1 to 10000
data: scala.collection.immutable.Range.Inclusive = Range(1, 2, 3, 4, 5, 6, 7, 8,
 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,
29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48,
49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68,
69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88,
89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 1
07, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 1
23, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 1
39, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 1
55, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170...

scala>
```

http://ht:4040/jobs/



2. Install Spark in centos Linux – 60 Minutes

Use: VM Centos 7 64 bit. CLI

Start the VM or the container and open a terminal to the host using putty. Perform the following activities in the console.

Download the required software

Apache Spark – Version 3.0.1 – Prebuilt for Apache Hadoop 3.2 and later.
File : spark-3.0.1-bin-hadoop3.2.tgz
Url : https://spark.apache.org/downloads.html

Inflate all the software in /opt folder.

JDK installation☹) (jdk-8u45-linux-x64.tar.gz)
https://www.oracle.com/in/java/technologies/javase/javase8-archive-downloads.html

Extract the jdk and rename the folder to jdk.

#tar -xvf jdk-8u45-linux-x64.tar.gz -C /opt
#mv jdk* jdk

Untar spark-X.X.0-bin-hadoop.X.tar to /opt

tar -xvf spark-x-bin-hadoop.x.tgz.gz -C /opt/

It should create a folder inside spark.X, rename to spark folder.

mv sparkX spark

Set the JAVA_HOME and initialize the PATH variable.

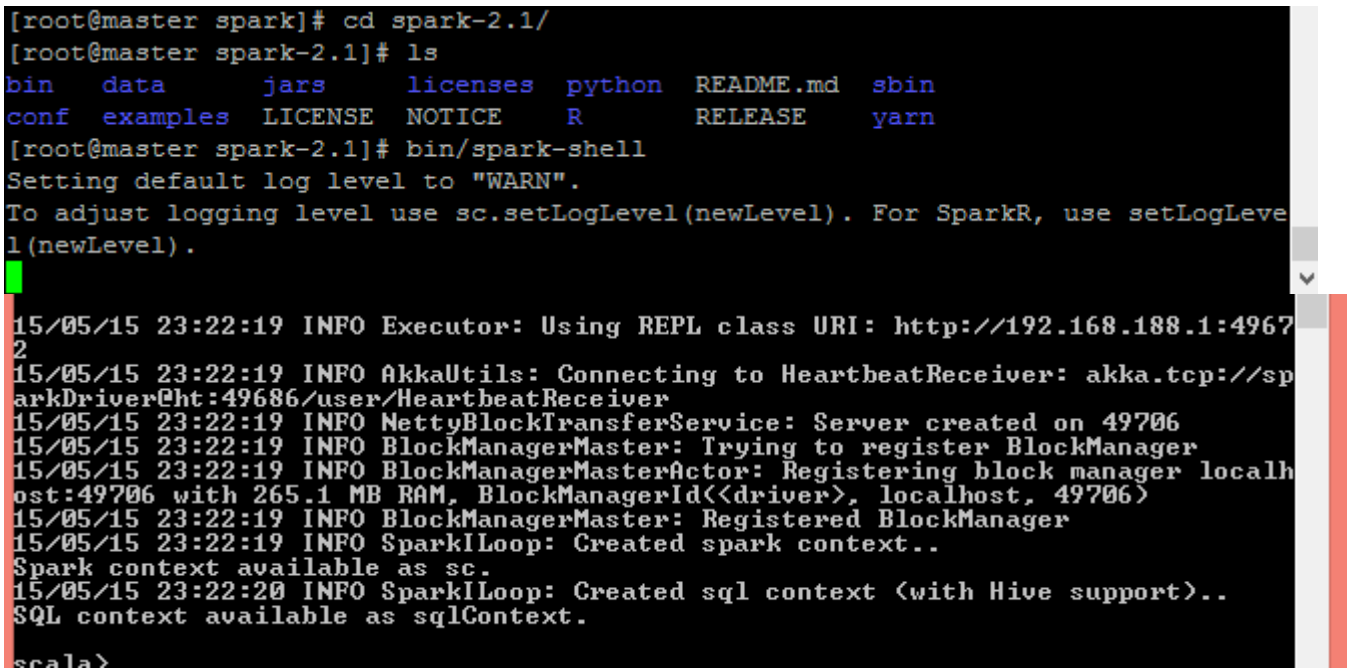
Open the profile and update with the following statements.

vi ~/.bashrc
export JAVA_HOME=/opt/jdk
export PATH=$PATH:$JAVA_HOME/bin
export PATH=$PATH:/opt/spark/bin

Type bash, to initialize the profile.

Go to the installation directory:

cd /opt/spark/bin
execute
./spark-shell



Enter the following command in the scala console to create a data set of 1...10 integers

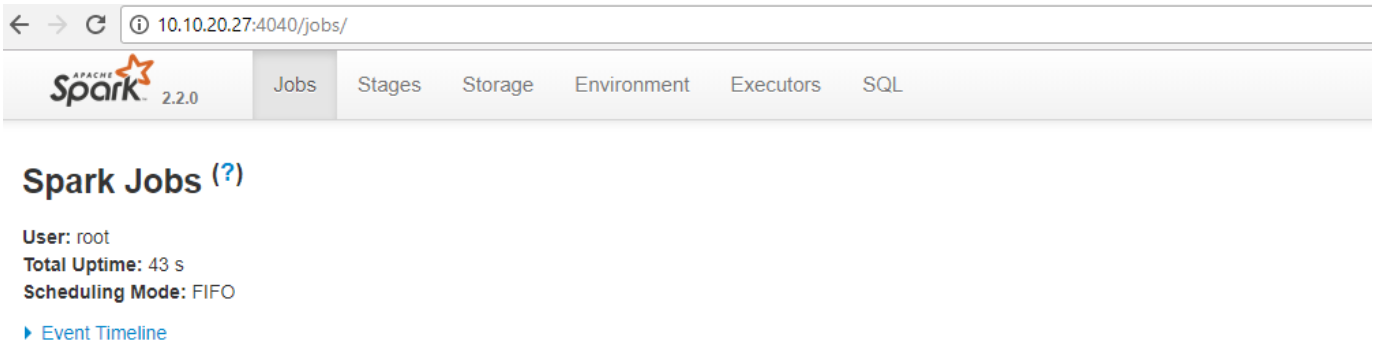
#val data = 1 to 10
#data.foreach(println)
#println(data(2))

```
scala> val data = 1 to 10
data: scala.collection.immutable.Range.Inclusive = Range 1 to 10

scala> data.foreach(println)
1
2
3
4
5
6
7
8
9
10

scala> println(data(2))
3

scala>
```

As seen above, you have initialized data variable with range from 1 to 10.
Then print its value using foreach method of scala.
You can also access the data using index.

Web UI of the spark shell can be accessed using the following URL.
You can click on each tab and verify the screen.

http://ht:4040/jobs/



Further details on this web UI will be discussed later.

You have successfully installed spark for local development.

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

3. Exploring DataFrames using the Apache Spark Shell – Scala – 35 Minutes

Following features of Spark will be demonstrated here:

- Loading json file.
- Understand its schema
- Select required fields.
- Apply filter.

Start spark-shell

#spark-shell
Create a text file users.json which contains sample data as listed below in data folder:

```
{"name":"Alice", "pcode":"94304"}
{"name":"Brayden", "age":30,
"pcode":"94304"}
{"name":"Carla", "age":19,
"pcode":"10036"}
{"name":"Diana", "age":46}
```

Scala:

Initiate the spark-shell from the folder which you have created the above file.

```
// Read the users json file as a dataframe.
val usersDF = spark.read.json("users.json")

// Find out the schema of the uploaded file
usersDF.printSchema()
```

As shown above, three fields will be displayed according to the json fields specified in the text file.

```
scala> val usersDF = spark.read.json("users.json")
usersDF: org.apache.spark.sql.DataFrame = [age: bigint, name: string ... 1 more field]

scala> usersDF.printSchema
root
 |-- age: long (nullable = true)
 |-- name: string (nullable = true)
 |-- pcode: string (nullable = true)


scala>
```

//Let us find out the first 3 records to have a sample data.
val users = usersDF.take(3)
usersDF.show()

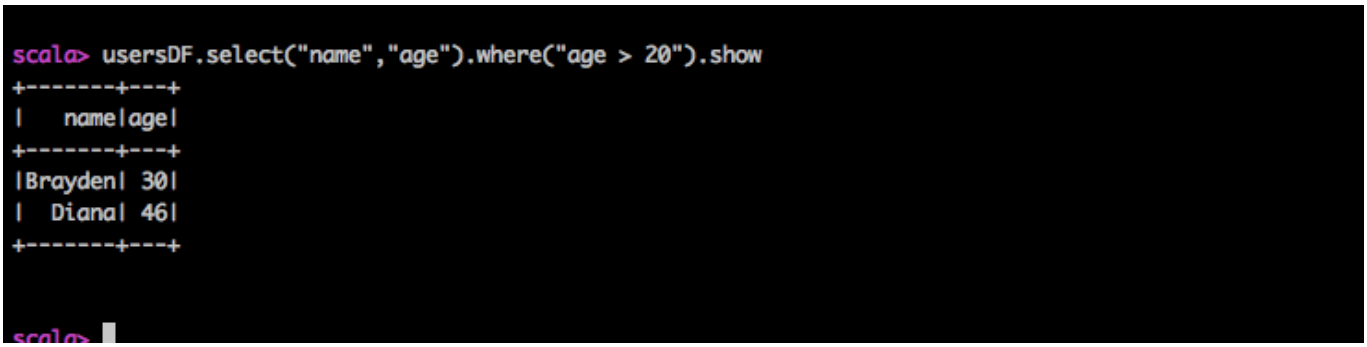
```
scala> val users = usersDF.take(3)
users: Array[org.apache.spark.sql.Row] = Array([null,Alice,94304], [30,Brayden,94304], [19,Carla,10036])

scala> usersDF.show()
+----+-------+-----+
| age|   name|pcode|
+----+-------+-----+
|null|  Alice|94304|
|  30|Brayden|94304|
|  19|  Carla|10036|
|  46|  Diana| null|
|null|Etienne|94104|
+----+-------+-----+


scala>
```

Out of the three fields, we are interested in only name and age fields. So, let us create a dataframe with only these two fields and apply a filter expression in which only person greater than 20 years are there in the dataframe.

val nameAgeDF = usersDF.select("name","age")
val nameAgeOver20DF = nameAgeDF.where("age > 20")
nameAgeOver20DF.show

```
scala> val nameAgeDF = usersDF.select("name","age")
nameAgeDF: org.apache.spark.sql.DataFrame = [name: string, age: bigint]

scala> val nameAgeOver20DF = nameAgeDF.where("age > 20")
nameAgeOver20DF: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [name: string, age: bigint]

scala> nameAgeOver20DF.show
+-------+---+
|   name|age|
+-------+---+
|Brayden| 30|
|  Diana| 46|
+-------+---+


scala>
```

usersDF.select("name","age").where("age > 20").show



You can also combine the functions as shown above. You will get the same result.

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

4. Working with DataFrames and Schemas – 30 Minutes

You will understand the following:

- Defining schema and mapping to dataframe while loading json file.
- Saving the dataframe to json file.

Create an input text file /software/people.csv

```
pcode,lastName,firstName,age
02134,Hopper,Grace,52
94020,Turing,Alan,32
94020,Lovelace,Ada,28
87501,Babbage,Charles,49
02134,Wirth,Niklaus,48
```

Execute the following in the Spark-shell.

Import and define the Structure.

```
import org.apache.spark.sql.types._
val columnsList = List(
        StructField("pcode", StringType),
        StructField("lastName", StringType),  StructField("firstName",
        StringType),  StructField("age", IntegerType))
val peopleSchema = StructType(columnsList)

// Specify the schema along with the loading instruction.
val usersDF  =
spark.read.option("header","true").schema(peopleSchema).csv("peop
le.csv")

usersDF.printSchema()
```

```
scala> val usersDF  =  spark.read.option("header","true").schema(peopleSchema).csv("people.csv")
usersDF: org.apache.spark.sql.DataFrame = [pcode: string, lastName: string ... 2 more fields]

scala> usersDF.printSchema()
root
 |-- pcode: string (nullable = true)
 |-- lastName: string (nullable = true)
 |-- firstName: string (nullable = true)
 |-- age: integer (nullable = true)
```

// As shown above, age is an Integer, which will be mapped to String by default in case its not define in the structure schema.

val nameAgeDF = usersDF.select("firstname","age")
nameAgeDF.show()

```
scala> val nameAgeDF =  usersDF.select("firstName","age")
nameAgeDF: org.apache.spark.sql.DataFrame = [firstName: string, age: int]

scala> nameAgeDF.show()
20/09/24 03:06:55 WARN CSVHeaderChecker: CSV header does not conform to the schema.
 Header: LastName, age
 Schema: firstName, age
Expected: firstName but found: LastName
CSV file: file:///software/people.csv
+---------+---+
|firstName|age|
+---------+---+
|    Grace| 52|
|     Alan| 32|
|      Ada| 28|
|  Charles| 49|
|  Niklaus| 48|
+---------+---+


scala>
```

You can save the dataframe consisting of First Name and age to a file.

nameAgeDF.write.json("age.json")

Open a terminal and verify the file. It will create a folder by the name age.json and inside that output will be there.

```
[root@306633508e8b software]# more age.json/

*** age.json/: directory ***

[root@306633508e8b software]# cd age.json/
[root@306633508e8b age.json]# ls
_SUCCESS  part-00000-8c5ff49a-9c1f-4294-b9c3-7c52c4b20147-c000.json
[root@306633508e8b age.json]# more part-00000-8c5ff49a-9c1f-4294-b9c3-7c52c4b20147-c000.json
{"firstName":"Grace","age":52}
{"firstName":"Alan","age":32}
{"firstName":"Ada","age":28}
{"firstName":"Charles","age":49}
{"firstName":"Niklaus","age":48}
[root@306633508e8b age.json]#
```

5. Use User Defined Functions & SQL

Load the data and convert into Dataframe.

```
val sfpdRDD = sc.textFile("sfpd.csv").map(_.split(","))

case class Incidents(incidentnum:String, category:String, description:String, dayofweek:String, date:String, time:String, pddistrict:String, resolution:String, address:String, X:Float, Y:Float, pdid:String)

val sfpdCase=sfpdRDD.map(inc=>Incidents(inc(0),inc(1), inc(2),inc(3),inc(4),inc(5),inc(6),inc(7),inc(8),inc(9).toFloat,inc(10).toFloat, inc(11)))

val sfpdDF=sfpdCase.toDF()
```

Register as a Table, so that we can perform SQL operation on it.

```
sfpdDF.registerTempTable("sfpd")
```

The date field in this dataset is a String of the form “mm/dd/yy”. We are going to create a function to extract the year from the date field. There are two ways to use user defined functions with Spark DataFrames. You can define it inline and use it within DataFrame operations or use it in SQL queries.

We can then find answers to questions such as: What is the number of incidents by year?

Define a function that extracts characters after the last ‘/’ from the string.

```
def getyear(s:String):String = {
    val year = s.substring(s.lastIndexOf('/')+1)
    year
}
```

Register the function

```
import spark.implicits._
```

spark.sqlContext.udf.register("getyear",getyear _)

Using the registered the udf in a SQL query, find the count of incidents by year.

val incyearSQL = spark.sqlContext.sql("SELECT getyear(date), count(incidentnum) AS countbyyear FROM sfpd GROUP BY getyear(date) ORDER BY countbyyear DESC")

incyearSQL.collect.foreach(println)

Find the category, address and resolution of incidents reported in 2014.

val inc2014 = spark.sqlContext.sql("SELECT category,address,resolution, date FROM sfpd WHERE getyear(date)='14'")

inc2014.collect.foreach(println)

Find the addresses and resolution of VANDALISM incidents in 2015.

val van2015 = spark.sqlContext.sql("SELECT category,address,resolution, date FROM sfpd WHERE getyear(date)='15' AND category='VANDALISM'")

van2015.collect.foreach(println)

van2015.count

SQL Engine:

In this mode, end-users or applications can interact with Spark SQL directly to run SQL queries, without the need to write any code.

To start the JDBC/ODBC server, run the following in the Spark directory:

```
./sbin/start-thriftserver.sh
```

Now you can use beeline to test the Thrift JDBC/ODBC server:

```
beeline
```

Connect to the JDBC/ODBC server in beeline with:

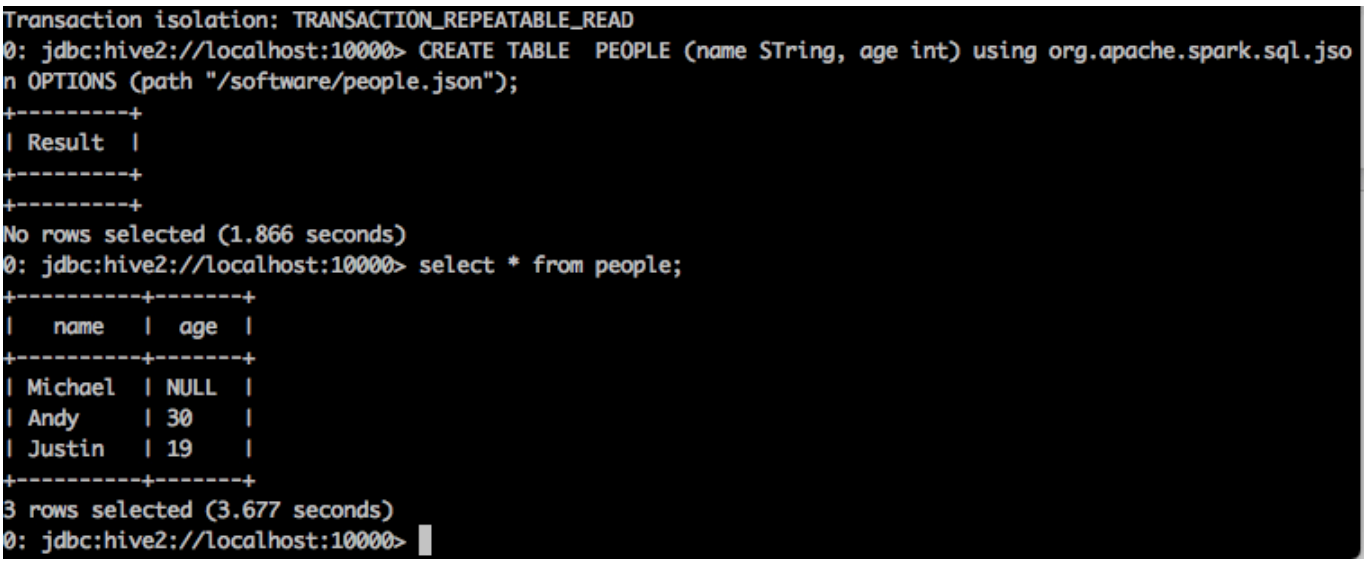
```
beeline> !connect jdbc:hive2://localhost:10000
```

It will ask for the userame, enter any say – henry leave the password blank.

Let us create a table and execute a query on it.

CREATE TABLE PEOPLE (name STring, age int) using org.apache.spark.sql.json OPTIONS (path "/software/people.json");

select * from people;



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

6. Analyzing Data with DataFrame Queries – 40 Minutes

We will understand the following in this lab:

- Join
- Accessing Columns in DF

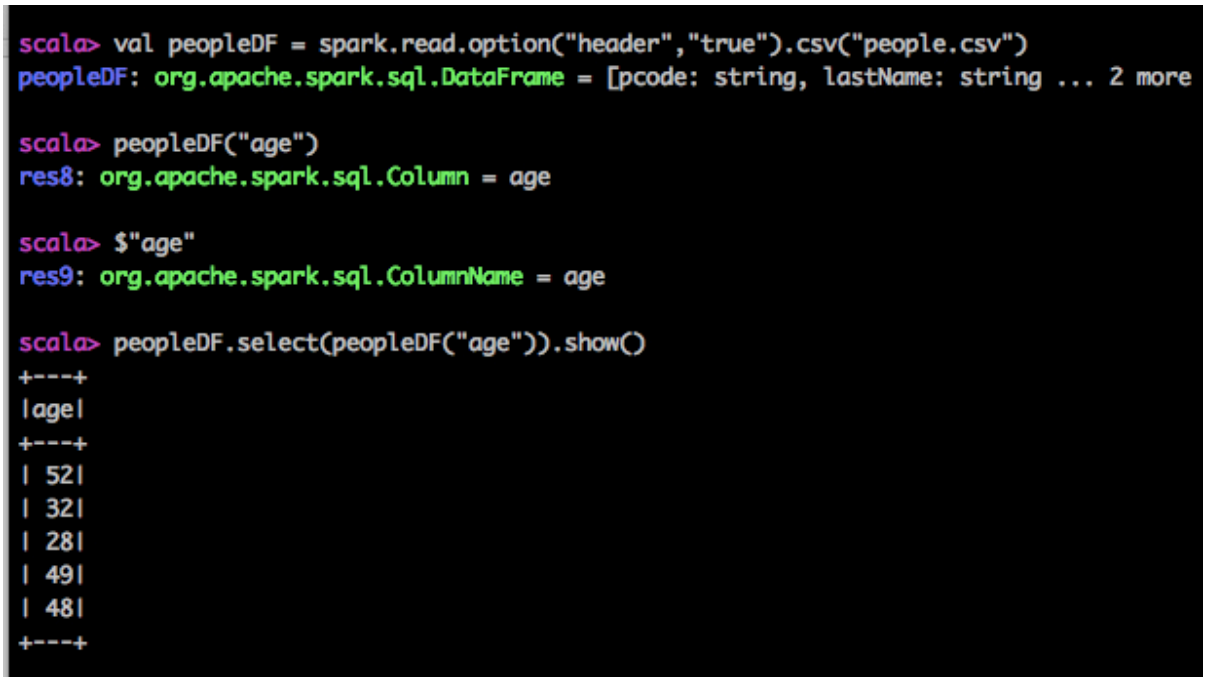
Use the following data for this lab:

/software/people.csv

pcode,lastName,firstNa
me,age
02134,Hopper,Grace,52
94020,Turing,Alan,32
94020,Lovelace,Ada,28

Load the people data and fetch column, age in the following way:

```
val peopleDF = spark.read.option("header","true").csv("people.csv")
peopleDF("age")
$"age"
peopleDF.select(peopleDF("age")).show()
```



Manipulate the column age i.e multiple age by 10.

peopleDF.select($"lastName", $"age" * 10).show

```
scala> peopleDF.select($"lastName", $"age" * 10).show
+--------+----------+
|lastName|(age * 10)|
+--------+----------+
|  Hopper|     520.0|
|  Turing|     320.0|
|Lovelace|     280.0|
| Babbage|     490.0|
|   Wirth|     480.0|
+--------+----------+
```

You can chain the Queries.

peopleDF.select($"lastName",($"age" * 10).alias("age_10")).show()

Perform aggregration

peopleDF.groupBy("pcode").count().show()

```
scala> peopleDF.select($"lastName",($"age" * 10).alias("age_10")).show()
+--------+------+
|lastName|age_10|
+--------+------+
|  Hopper| 520.0|
|  Turing| 320.0|
|Lovelace| 280.0|
| Babbage| 490.0|
|   Wirth| 480.0|
+--------+------+


scala> peopleDF.groupBy("pcode").count().show()
+-----+-----+
|pcode|count|
+-----+-----+
|87501|    1|
|94020|    2|
|02134|    2|
+-----+-----+
```

Next let us join two dataframes:

/software/people-no-pcode.csv

```
pcode,lastName,firstName,age
02134,Hopper,Grace,52
,Turing,Alan,32
94020,Lovelace,Ada,28
87501,Babbage,Charles,49
02134,Wirth,Niklaus,48
```

/software/pcodes.csv

```
pcode,city,state
02134,Boston,MA
94020,Palo Alto,CA
87501,Santa Fe,NM
60645,Chicago,IL
```

Load the people and code files in DF.

```
val peopleDF = spark.read.option("header","true").csv("people-no-pcode.csv")
val pcodesDF = spark.read.option("header","true").csv("pcodes.csv")
```

Peform Inner Join on pcode.

```
peopleDF.join(pcodesDF, "pcode").show()
```

```
scala> val peopleDF = spark.read.option("header","true").csv("people-no-pcode.csv")
peopleDF: org.apache.spark.sql.DataFrame = [pcode: string, lastName: string ... 2 more fields]

scala> peopleDF.join(pcodesDF, "pcode").show()
+-----+--------+---------+---+---------+-----+
|pcode|lastName|firstName|age|     city|state|
+-----+--------+---------+---+---------+-----+
|02134|  Hopper|    Grace| 52|   Boston|   MA|
|94020|Lovelace|      Ada| 28|Palo Alto|   CA|
|87501| Babbage|  Charles| 49| Santa Fe|   NM|
|02134|   Wirth|  Niklaus| 48|   Boston|   MA|
+-----+--------+---------+---+---------+-----+


scala>
```

Perform the Left outer join

peopleDF.join(pcodesDF,peopleDF("pcode") ===
pcodesDF("pcode"), "left_outer").show

You can see null value in the pcode of the second row.

Joining on Columns with Different Names

/software/zcodes.csv

zip,city,state
02134,Boston,MA
94020,Palo Alto,CA
87501,Santa Fe,NM

Join with the pcode and zip of the second file.

```
val zcodesDF = spark.read.option("header","true").csv("zcodes.csv")
peopleDF.join(zcodesDF, $"pcode" === $"zip").show
```

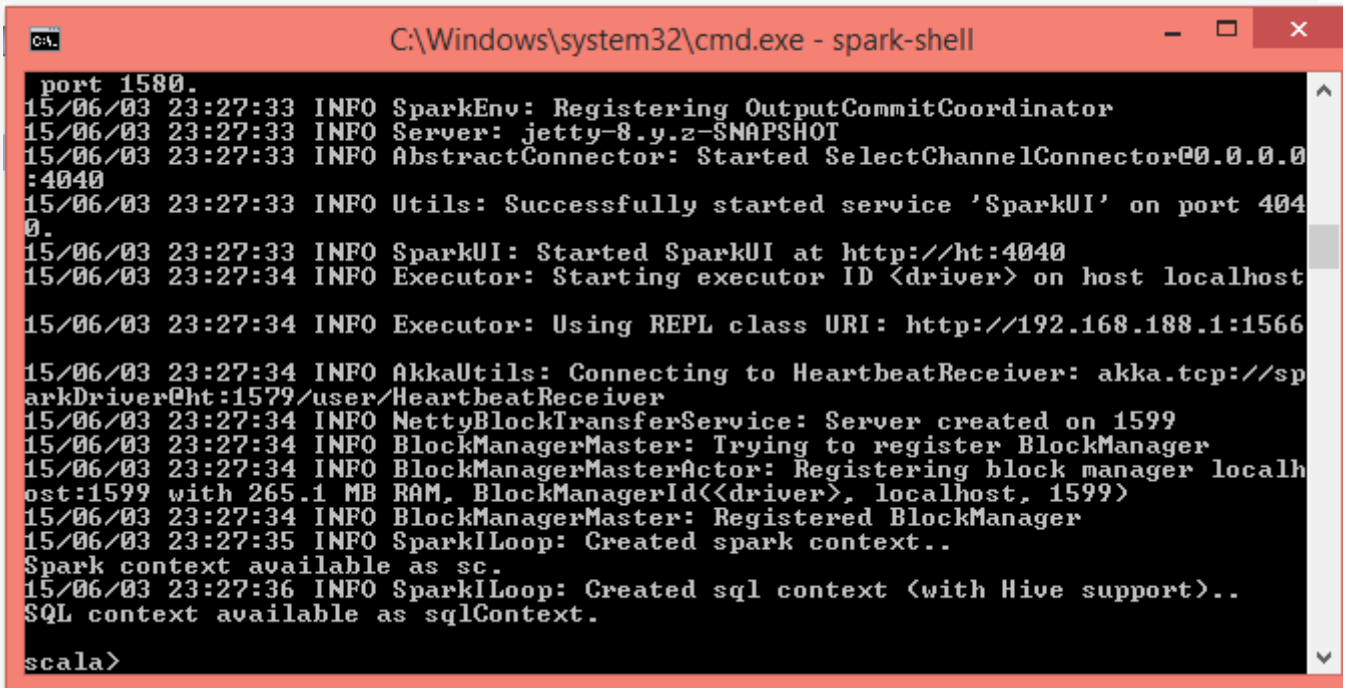
```
scala> peopleDF.join(zcodesDF, $"pcode" === $"zip").show
+-----+--------+---------+---+-----+---------+-----+
|pcode|lastName|firstName|age|  zip|     city|state|
+-----+--------+---------+---+-----+---------+-----+
|02134|  Hopper|    Grace| 52|02134|   Boston|   MA|
|94020|Lovelace|      Ada| 28|94020|Palo Alto|   CA|
|87501| Babbage|  Charles| 49|87501| Santa Fe|   NM|
|02134|   Wirth|  Niklaus| 48|02134|   Boston|   MA|
+-----+--------+---------+---+-----+---------+-----+
```

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

7. Interactive RDD Analysis with the Spark Shell – 30 Minutes

Start the spark shell:

```
spark-shell
```



Running word count:
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._

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

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

// File is in spark installation folder. You can copy it to /software
val txtFile = "README.md"
val txtData = sc.textFile(txtFile)
txtData.cache()

```
scala> val txtFile = "spam.data"
txtFile: String = spam.data

scala> val txtData = sc.textFile(txtFile)
15/05/16 16:33:21 INFO MemoryStore: ensureFreeSpace(159214) called with curMem=1
81810, maxMem=278019440
15/05/16 16:33:21 INFO MemoryStore: Block broadcast_1 stored as values in memory
 (estimated size 155.5 KB, free 264.8 MB)
15/05/16 16:33:21 INFO MemoryStore: ensureFreeSpace(22692) called with curMem=34
1024, maxMem=278019440
15/05/16 16:33:21 INFO MemoryStore: Block broadcast_1_piece0 stored as bytes in
memory (estimated size 22.2 KB, free 264.8 MB)
15/05/16 16:33:21 INFO BlockManagerInfo: Added broadcast_1_piece0 in memory on l
ocalhost:63788 (size: 22.2 KB, free: 265.1 MB)
15/05/16 16:33:21 INFO BlockManagerMaster: Updated info of block broadcast_1_pie
ce0
15/05/16 16:33:21 INFO SparkContext: Created broadcast 1 from textFile at <conso
le>:27
txtData: org.apache.spark.rdd.RDD[String] = spam.data MapPartitionsRDD[3] at tex
tFile at <console>:27

scala> txtData.cache()
res2: txtData.type = spam.data MapPartitionsRDD[3] at textFile at <console>:27
```

txtData.count()

```
scala> txtData.count()
15/05/16 16:33:43 INFO FileInputFormat: Total input paths to process : 1
15/05/16 16:33:43 INFO SparkContext: Starting job: count at <console>:30
15/05/16 16:33:43 INFO DAGScheduler: Got job 0 (count at <console>:30) with 2 ou
tput partitions (allowLocal=false)
15/05/16 16:33:43 INFO DAGScheduler: Final stage: Stage 0(count at <console>:30)

15/05/16 16:33:43 INFO DAGScheduler: Parents of final stage: List()
15/05/16 16:33:43 INFO DAGScheduler: Missing parents: List()
15/05/16 16:33:43 INFO DAGScheduler: Submitting Stage 0 (spam.data MapPartitions
RDD[3] at textFile at <console>:27), which has no missing parents
15/05/16 16:33:43 INFO MemoryStore: ensureFreeSpace(2640) called with curMem=363
716, maxMem=278019440
15/05/16 16:33:43 INFO MemoryStore: Block broadcast_2 stored as values in memory
 (estimated size 2.6 KB, free 264.8 MB)
15/05/16 16:33:43 INFO MemoryStore: ensureFreeSpace(1929) called with curMem=366
356, maxMem=278019440
15/05/16 16:33:43 INFO MemoryStore: Block broadcast_2_piece0 stored as bytes in
memory (estimated size 1929.0 B, free 264.8 MB)
15/05/16 16:33:43 INFO BlockManagerInfo: Added broadcast_2_piece0 in memory on l
ocalhost:63788 (size: 1929.0 B, free: 265.1 MB)
15/05/16 16:33:43 INFO BlockManagerMaster: Updated info of block broadcast_2_pie
ce0
15/05/16 16:33:43 INFO SparkContext: Created broadcast 2 from broadcast at DAGSc
heduler.scala:839
15/05/16 16:33:44 INFO DAGScheduler: Submitting 2 missing tasks from Stage 0 (sp
am.data MapPartitionsRDD[3] at textFile at <console>:27)
15/05/16 16:33:44 INFO TaskSchedulerImpl: Adding task set 0.0 with 2 tasks
15/05/16 16:33:44 INFO TaskSetManager: Starting task 0.0 in stage 0.0 (TID 0, lo
calhost, PROCESS_LOCAL, 1309 bytes)
15/05/16 16:33:44 INFO TaskSetManager: Starting task 1.0 in stage 0.0 (TID 1, lo
calhost, PROCESS_LOCAL, 1309 bytes)
15/05/16 16:33:44 INFO Executor: Running task 0.0 in stage 0.0 (TID 0)
15/05/16 16:33:44 INFO Executor: Running task 1.0 in stage 0.0 (TID 1)
15/05/16 16:33:44 INFO CacheManager: Partition rdd_3_1 not found, computing it
15/05/16 16:33:44 INFO CacheManager: Partition rdd_3_0 not found, computing it
15/05/16 16:33:44 INFO HadoopRDD: Input split: file:/D:/MyExperiment/spark-1.3/b
in/spam.data:0+349170
```

Output:

```
15/05/16 16:33:45 INFO DAGScheduler: Job 0 finished: count at <console>:30, took
 1.728775 s
res3: Long = 4601

scala>
```

Create RDD from Collection.

```
val myData = Seq("Alice","Carlos","Frank","Barbara")
val myRDD = sc.parallelize(myData)
for (line <- myRDD.take(2))  println(line)
myRDD.saveAsTextFile("mydata/")
```

```
scala> val myData = Seq("Alice","Carlos","Frank","Barbara")
myData: Seq[String] = List(Alice, Carlos, Frank, Barbara)

scala> val myRDD = sc.parallelize(myData)
myRDD: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[120] at parallelize at <console>:30

scala> for (line <- myRDD.take(2))  println(line)
Alice
Carlos

scala> myRDD.saveAsTextFile("mydata/")
```

Verify the output.
Go to /software folder. You will find a folder mydata. Inside it, two files starting with part* will be created. You can open the file to view the output.

```
[root@893094f7bb92 software]# ls -lt
total 63708
drwxr-xr-x 2 root root      4096 Nov  4 10:54 mydata
-rw-r--r-- 1 root root        85 Nov  4 09:15 zcodes.csv
-rw-r--r-- 1 root root       138 Nov  4 09:11 people-no-pcode.csv
-rw-r--r-- 1 root root        87 Nov  4 09:04 pcodes.csv
-rw-r--r-- 1 root root       143 Nov  4 08:48 people.csv
-rw-r--r-- 1  501 games  2047667 Nov  3 10:23 sfpd.csv
drwxr-xr-x 2 root root      4096 Nov  3 10:10 age1.json
drwxr-xr-x 2 root root      4096 Nov  3 10:05 age.json
-rw-r--r-- 1 root root       188 Nov  3 09:55 users.json
-rw-r--r-- 1 root root      4488 Sep 10 09:54 README.md
-rw-r--r-- 1 root root  63147059 Aug 14 15:38 influxdb-1.8.2.x86_64.rpm
[root@893094f7bb92 software]# more mydata/part-00000
Alice
Carlos
[root@893094f7bb92 software]# more mydata/part-00001
Frank
Barbara
[root@893094f7bb92 software]#
```

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

8. Transformation and Action – RDD – 45 Minutes

Creata a java class RDDTransformation.java

Let's make a new RDD from the text of the README file in the Spark source directory: (SPARK_HOME/README.md)

```
scala> val textFile = sc.textFile("README.md")
```

```
scala> val textFile  = sc.textFile("/MyTrainingWork/Spark/README.md")
15/06/03 23:32:06 INFO MemoryStore: ensureFreeSpace(73391) called with curMem=18
1810, maxMem=278019440
15/06/03 23:32:06 INFO MemoryStore: Block broadcast_1 stored as values in memory
 (estimated size 71.7 KB, free 264.9 MB)
15/06/03 23:32:06 INFO MemoryStore: ensureFreeSpace(31262) called with curMem=25
5201, maxMem=278019440
15/06/03 23:32:06 INFO MemoryStore: Block broadcast_1_piece0 stored as bytes in
memory (estimated size 30.5 KB, free 264.9 MB)
15/06/03 23:32:06 INFO BlockManagerInfo: Added broadcast_1_piece0 in memory on l
ocalhost:1599 (size: 30.5 KB, free: 265.1 MB)
15/06/03 23:32:06 INFO BlockManagerMaster: Updated info of block broadcast_1_pie
ce0
15/06/03 23:32:06 INFO SparkContext: Created broadcast 1 from textFile at <conso
le>:21
textFile: org.apache.spark.rdd.RDD[String] = /MyTrainingWork/Spark/README.md Map
PartitionsRDD[3] at textFile at <console>:21
```

Let's start with a few actions:

```
scala> textFile.count() // Number of items in this RDD
```

```
scala> textFile.count()
15/06/03 23:32:16 INFO FileInputFormat: Total input paths to process : 1
15/06/03 23:32:16 INFO SparkContext: Starting job: count at <console>:24
15/06/03 23:32:16 INFO DAGScheduler: Got job 0 (count at <console>:24) with 2 ou
tput partitions (allowLocal=false)
15/06/03 23:32:16 INFO DAGScheduler: Final stage: Stage 0(count at <console>:24)

15/06/03 23:32:16 INFO DAGScheduler: Parents of final stage: List()
15/06/03 23:32:16 INFO DAGScheduler: Missing parents: List()
15/06/03 23:32:16 INFO DAGScheduler: Submitting Stage 0 (/MyTrainingWork/Spark/R
EADME.md MapPartitionsRDD[3] at textFile at <console>:21), which has no missing
parents
```

```
15/06/03 23:32:17 INFO TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in
 352 ms on localhost (1/2)
15/06/03 23:32:17 INFO TaskSetManager: Finished task 1.0 in stage 0.0 (TID 1) in
 338 ms on localhost (2/2)
15/06/03 23:32:17 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have
all completed, from pool
15/06/03 23:32:17 INFO DAGScheduler: Stage 0 (count at <console>:24) finished in
 0.393 s
15/06/03 23:32:17 INFO DAGScheduler: Job 0 finished: count at <console>:24, took
 0.678076 s
res2: Long = 98

scala>
```

```
scala> textFile.first() // First item in this RDD
```

```
.md:0+1814
15/06/03 23:34:06 INFO Executor: Finished task 0.0 in stage 1.0 (TID 2). 1809 by
tes result sent to driver
15/06/03 23:34:06 INFO TaskSetManager: Finished task 0.0 in stage 1.0 (TID 2) in
 10 ms on localhost (1/1)
15/06/03 23:34:06 INFO DAGScheduler: Stage 1 (first at <console>:24) finished in
 0.011 s
15/06/03 23:34:06 INFO TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks have
all completed, from pool
15/06/03 23:34:06 INFO DAGScheduler: Job 1 finished: first at <console>:24, took
 0.019875 s
res3: String = # Apache Spark

scala>
```

Now let's use a transformation. We will use the filter transformation to return a new RDD with a subset of the items in the file.

```
scala> val linesWithSpark = textFile.filter(line => line.contains("Spark"))
```

We can chain together transformations and actions:

```
scala> textFile.filter(line => line.contains("Spark")).count() // How many lines
 contain "Spark"?
```

```
15/06/03 23:35:17 INFO Executor: Finished task 0.0 in stage 2.0 (TID 3). 1830 by
tes result sent to driver
15/06/03 23:35:17 INFO TaskSetManager: Finished task 1.0 in stage 2.0 (TID 4) in
 12 ms on localhost (1/2)
15/06/03 23:35:17 INFO TaskSetManager: Finished task 0.0 in stage 2.0 (TID 3) in
 14 ms on localhost (2/2)
15/06/03 23:35:17 INFO DAGScheduler: Stage 2 (count at <console>:24) finished in
 0.014 s
15/06/03 23:35:17 INFO TaskSchedulerImpl: Removed TaskSet 2.0, whose tasks have
all completed, from pool
15/06/03 23:35:17 INFO DAGScheduler: Job 2 finished: count at <console>:24, took
 0.028214 s
res4: Long = 19

scala>
```

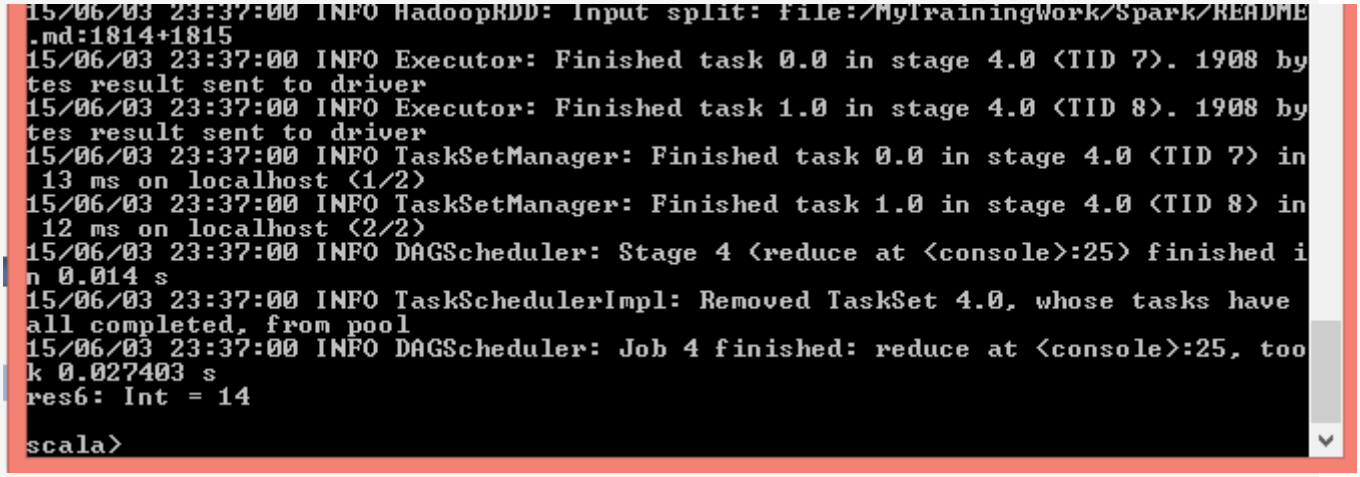
Let's say we want to find the line with the most words

```
scala> textFile.map(line => line.split(" ").size).reduce((a, b) => if (a > b) a els
e b)
```

```
calhost, PROCESS_LOCAL, 1300 bytes)
15/06/03 23:36:03 INFO Executor: Running task 0.0 in stage 3.0 (TID 5)
15/06/03 23:36:03 INFO Executor: Running task 1.0 in stage 3.0 (TID 6)
15/06/03 23:36:03 INFO HadoopRDD: Input split: file:/MyTrainingWork/Spark/README
.md:0+1814
15/06/03 23:36:03 INFO HadoopRDD: Input split: file:/MyTrainingWork/Spark/README
.md:1814+1815
15/06/03 23:36:03 INFO Executor: Finished task 1.0 in stage 3.0 (TID 6). 1908 by
tes result sent to driver
15/06/03 23:36:03 INFO Executor: Finished task 0.0 in stage 3.0 (TID 5). 1908 by
tes result sent to driver
15/06/03 23:36:03 INFO TaskSetManager: Finished task 1.0 in stage 3.0 (TID 6) in
 15 ms on localhost (1/2)
15/06/03 23:36:03 INFO TaskSetManager: Finished task 0.0 in stage 3.0 (TID 5) in
 17 ms on localhost (2/2)
15/06/03 23:36:03 INFO DAGScheduler: Stage 3 (reduce at <console>:24) finished i
n 0.019 s
15/06/03 23:36:03 INFO TaskSchedulerImpl: Removed TaskSet 3.0, whose tasks have
all completed, from pool
15/06/03 23:36:03 INFO DAGScheduler: Job 3 finished: reduce at <console>:24, too
k 0.033852 s
res5: Int = 14

scala>
```

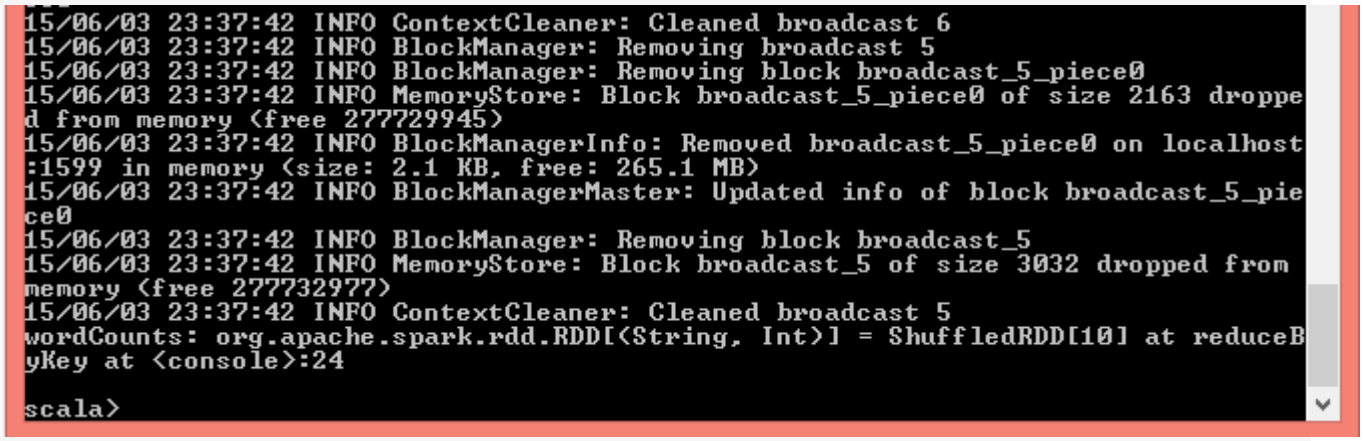
we can easily call functions declared elsewhere. We'll use Math.max() function to make this code easier to understand:

```
scala> import java.lang.Math
scala> textFile.map(line => line.split(" ").size).reduce((a, b) => Math.max(a, b
))
```



One common data flow pattern is MapReduce, as popularized by Hadoop. Spark can implement MapReduce flows easily:

```
scala> val wordCounts = textFile.flatMap(line => line.split(" ")).map(word =>
(word, 1)).reduceByKey((a, b) => a + b)
```



Here, we combined the flatMap, map and reduceByKey transformations to compute the per-word counts in the file as an RDD of (String, Int) pairs. To collect the word counts in our shell, we can use the collect action:

```
scala> wordCounts.collect()
```

let's mark our linesWithSpark dataset to be cached:

```
15/06/03 23:39:02 INFO TaskSchedulerImpl: Removed TaskSet 6.0, whose tasks have
all completed, from pool
15/06/03 23:39:02 INFO DAGScheduler: Job 5 finished: collect at <console>:27, to
ok 0.346996 s
res7: Array[(String, Int)] = Array((package,1), (this,1), (Because,1), (Python,2
), (cluster.,1), (its,1), ([run,1), (general,2), (YARN,,1), (have,1), (pre-built
,1), (locally.,1), (changed,1), (locally,2), (sc.parallelize(1,1), (only,1), (se
veral,1), (This,2), (basic,1), (first,1), (documentation,3), (Configuration,1),
(learning,,1), (graph,1), (Hive,2), (["Specifying,1), ("yarn-client",1), (page](
http://spark.apache.org/documentation.html),1), ([params]`.,1), (application,1),
 ([project,2), (prefer,1), (SparkPi,2), ((<http://spark.apache.org/>),1), (engine,
1), (version,1), (file,1), (documentation.,1), (MASTER,1), (example,3), (are,1),
 (systems.,1), (params,1), (scala>,1), (provides,1), (refer,2), (configure,1), (
Interactive,2), (distribution.,1), (can,6), (build,3), (when,1), (Apache,1), ...

scala>
```

```
scala> linesWithSpark.cache()
scala> linesWithSpark.count()
```

```
all completed, from pool
15/06/03 23:39:46 INFO DAGScheduler: Stage 7 (count at <console>:27) finished in
 0.056 s
15/06/03 23:39:46 INFO DAGScheduler: Job 6 finished: count at <console>:27, took
 0.070778 s
res9: Long = 19

scala>
```

Next, we will convert RDD to Dataframe.

Create a file /software/people.txt

```
02134,Hopper,Grace,5
2
94020,Turing,Alan,32
94020,Lovelace,Ada,3
```

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

val mySchema = StructType(Array(  StructField("pcode",
StringType),  StructField("lastName", StringType),
StructField("firstName", StringType),  StructField("age",
IntegerType)
))

val rowRDD = sc.textFile("people.txt").map(line =>
line.split(",")).map(values =>
Row(values(0),values(1),values(2),values(3).toInt))

val myDF = spark.createDataFrame(rowRDD,mySchema)

myDF.show(2)
```

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

scala>  import org.apache.spark.sql.Row
import org.apache.spark.sql.Row

scala> val mySchema = StructType(Array(  StructField("pcode", StringType),  StructField("lastName", StringType),  Struct
Field("firstName", StringType),  StructField("age", IntegerType)
     | ))
mySchema: org.apache.spark.sql.types.StructType = StructType(StructField(pcode,StringType,true), StructField(lastName,St
ringType,true), StructField(firstName,StringType,true), StructField(age,IntegerType,true))

scala> val rowRDD = sc.textFile("people.txt").map(line => line.split(",")).map(values =>
     | Row(values(0),values(1),values(2),values(3).toInt))
rowRDD: org.apache.spark.rdd.RDD[org.apache.spark.sql.Row] = MapPartitionsRDD[12] at map at <console>:29

scala>

scala> val myDF = spark.createDataFrame(rowRDD,mySchema)
myDF: org.apache.spark.sql.DataFrame = [pcode: string, lastName: string ... 2 more fields]

scala> myDF.show(2)
```

Profile: Default
Command: /Users/henrypotsangbam/Applications/iTerm.app/Contents/MacOS/iTerm2

```
scala> myDF.show(2)
+-----+--------+---------+---+
|pcode|lastName|firstName|age|
+-----+--------+---------+---+
|02134|  Hopper|    Grace| 52|
|94020|  Turing|     Alan| 32|
+-----+--------+---------+---+
only showing top 2 rows
```

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

9. Work with Pair RDD – 30 Minutes

Lab Overview

In this activity, you will load SFPD data from a CSV file. You will create pair RDD and apply pair RDD operations to explore the data.

Scenario

Our dataset is a .csv file that consists of SFPD incident data from SF OpenData (https://data.sfgov.org/). For each incident, we have the following information:

Field	Description	Example Value
IncidentNum	Incident number	150561637
Category	Category of incident	NON-CRIMINAL
Descript	Description of incident	FOUND_PROPERTY
DayOfWeek	Day of week that incident occurred	Sunday
Date	Date of incident	6/28/15
Time	Time of incident	23:50
PdDistrict	Police Department District	TARAVAL

Resolution	Resolution	NONE
Address	Address	1300_Block_of_LA_PLAYA_ST
X	X-coordinate of location	-122.5091348
Y	Y-coordinate of location	37.76119777
PdID	Department ID	15056163704013

The dataset has been modified to decrease the size of the files and also to make it easier to use. We use this same dataset for all the labs in this course.

Lab 4.1: Load Data into Apache Spark

Objectives

- Launch the Spark interactive shell
- Load data into Spark
- Explore data in Apache Spark

4.1.1 Launch the Spark Interactive Shell

The Spark interactive shell is available in Scala or Python.

Note: All instructions here are for Scala.

1. To launch the Interactive Shell, at the command line, run the following command:

```
spark-shell --master local[2]
```

Note: To quit the Scala Interactive shell, use the command

```
Exit
```

4.1.2. Load Data into Spark

(copy all data files in sparkcluster)

To load the data we are going to use the `SparkContext` method `textFile`. The SparkContext is available in the interactive shell as the variable **`sc`**. We also want to split the file by the separator ",".

1. We define the mapping for our input variables. While this isn't a necessary step, it makes it easier to refer to the different fields by names.

```
val IncidntNum = 0
val Category = 1
val Descript = 2
val DayOfWeek = 3
val Date = 4
val Time = 5
val PdDistrict = 6
val Resolution = 7
val Address = 8
val X = 9
val Y = 10
val PdId = 11
```

2. To load data into Spark, at the Scala command prompt:

```
val sfpdRDD =sc.textFile("sfpd.csv").map(_.split(","))
```

Lab 4.1.3 Explore data using RDD operations

What transformations and actions would you use in each case? Complete the command with the appropriate transformations and actions.

1. How do you see the first element of the inputRDD (sfpdRDD)?

```
sfpdRDD.first()
```

```
scala> sfpdRDD.first()
res4: Array[String] = Array(150599321, OTHER_OFFENSES, POSSESSION_OF_BURGLARY_TO
OLS, Thursday, 7/9/15, 23:45, CENTRAL, ARREST/BOOKED, JACKSON_ST/POWELL_ST, -122
.4099006, 37.79561712, 15059900000000)
```

2. What do you use to see the first 5 elements of the RDD?

sfpdRDD.take(5)

```
scala> sfpdRDD.take(5)
res6: Array[Array[String]] = Array(Array(150599321, OTHER_OFFENSES, POSSESSION_O
F_BURGLARY_TOOLS, Thursday, 7/9/15, 23:45, CENTRAL, ARREST/BOOKED, JACKSON_ST/PO
WELL_ST, -122.4099006, 37.79561712, 15059900000000), Array(156168837, LARCENY/TH
EFT, PETTY_THEFT_OF_PROPERTY, Thursday, 7/9/15, 23:45, CENTRAL, NONE, 300_Block_
of_POWELL_ST, -122.4083843, 37.78782711, 15616900000000), Array(150599321, OTHER
_OFFENSES, DRIVERS_LICENSE/SUSPENDED_OR_REVOKED, Thursday, 7/9/15, 23:45, CENTRA
L, ARREST/BOOKED, JACKSON_ST/POWELL_ST, -122.4099006, 37.79561712, 1505990000000
0), Array(150599224, OTHER_OFFENSES, DRIVERS_LICENSE/SUSPENDED_OR_REVOKED, Thurs
day, 7/9/15, 23:36, PARK, ARREST/BOOKED, MASONIC_AV/GOLDEN_GATE_AV, -122.4468469
, 37.77766882, 15059900000000), Array(156169067, LARCENY/THEFT, GRAND_THEFT_F...
```

3. What is the total number of incidents?

val totincs = sfpdRDD.count()

```
scala> sfpdRDD.count()
res8: Long = 383775
```

4. What is the total number of distinct resolutions?

val totres = sfpdRDD.map(inc=>inc(7)).distinct.count()

```
scala> sfpdRDD.map(inc=>inc(7)).distinct.count()
res11: Long = 17
```

5. List the distinct PdDistricts.

val districts =`sfpdRDD.map(inc=>inc(6)).collect()`

```
scala> sfpdRDD.map(inc=>inc(6)).distinct.collect()
res14: Array[String] = Array(INGLESIDE, SOUTHERN, PARK, NORTHERN, MISSION, RICHM
OND, TENDERLOIN, BAYVIEW, TARAVAL, CENTRAL)
```

10. Create & Explore PairRDD – 60 Minutes

In the previous activity we explored the data in the sfpdRDD. We used RDD operations. In this activity, we will create pairRDD to find answers to questions about the data.

Objectives

- Create pairRDD & apply pairRDD operations
- Join pairRDD

Create pair RDD & apply pair RDD operations

Start the spark shell if not done else skip the next command.

#spark-shell

val sfpdRDD = sc.textFile("sfpd.csv").map(line=>line.split(","))

1. Which five districts have the highest incidents?

Note: This is similar to doing a word count. First, use the `map` transformation to create a pair RDD with the key being the field on which you want to count. In this case, the key would be PdDistrict and the value is "1" for each count of the district.

To get the total count, use `reduceByKey((a,b)=>a+b)`.

To find the **top** 5, you have to do a sort in descending. However, sorting on the result of the reduceByKey will sort on the District rather than the count. Apply the `map` transformation to create a pairRDD with the count being the key and the value being the district.

Then use `sortByKey`(false) to specify descending order. To get the top 5, use `take(5)`.

Note that this is one way of doing it. There may be other ways of achieving the same result.

Step to be follow (Hints)

a. Use a `map` transformation to create pair RDD from sfpdRDD of the form: [(PdDistrict, 1)]

b. Use `reduceByKey((a,b)=>a+b)` to get the count of incidents in each district. Your result will be a pairRDD of the form [(PdDistrict, count)]

c. Use map again to get a pairRDD of the form [(count, PdDistrict)]

d. Use `sortByKey(false)` on [(count, PdDistrict)]

e. Use `take(5)` to get the top 5.

Answer:

```
val top5Dists = sfpdRDD.map(incident=>(incident(6),1)).reduceByKey((x,y)
=>x+y).map(x=>(x._2,x._1)).sortByKey(false).take(3)
```

```
scala> val top5Dists = sfpdRDD.map(incident=>(incident(6),1)).reduceByKey((x,y)
     | =>x+y).map(x=>(x._2,x._1)) .sortByKey(false).take(3)
top5Dists: Array[(Int, String)] = Array((73308,SOUTHERN), (50164,MISSION), (4687
7,NORTHERN))
```

2. Which five addresses have the highest incidents?
   a. Create pairRDD (map)
   b. Get the count for key (reduceByKey)
   c. Pair RDD with key and count switched (map
   d. Sort in descending order (sortByKey
   e. Use **`take(5)`** to get the top 5

   Answer:

```
val top5Adds = sfpdRDD.map(incident=>(incident(8),1)).reduceByKey((x,y)=>x
+y).map(x=>(x._2,x._1)).sortByKey(false).take(5)
```

```
scala> val top5Adds = sfpdRDD.map(incident=>(incident(8),1)).reduceByKey((x,y)=>
x
     | +y).map(x=>(x._2,x._1)).sortByKey(false).take(5)
top5Adds: Array[(Int, String)] = Array((10852,800_Block_of_BRYANT_ST), (3671,800
_Block_of_MARKET_ST), (2027,1000_Block_of_POTRERO_AV), (1585,2000_Block_of_MISSI
ON_ST), (1512,16TH_ST/MISSION_ST))

scala>
```

3. What are the top **three** categories of incidents?

Answer:
val top3Cat = sfpdRDD.map(incident=>(incident(1),1)).reduceByKey((x,y)=>
x+y).map(x=>(x._2,x._1)) .sortByKey(false).take(3)

```
scala> val top3Cat = sfpdRDD.map(incident=>(incident(1),1)).reduceByKey((x,y)=>
x+y).map(x=>(x._2,x._1)) .sortByKey(false).take(3)
top3Cat: Array[(Int, String)] = Array((96955,LARCENY/THEFT), (50611,OTHER_OFFENS
ES), (50269,NON-CRIMINAL))
```

4. . What is the count of incidents by district?

Answer:
val num_inc_dist = sfpdRDD.map(incident=>(incident(6),1)).countByKey()

```
scala> val num_inc_dist = sfpdRDD.map(incident=>(incident(6),1)).countByKey()
num_inc_dist: scala.collection.Map[String,Long] = Map(SOUTHERN -> 73308, INGLESI
DE -> 33159, TENDERLOIN -> 30174, MISSION -> 50164, TARAVAL -> 27470, RICHMOND -
> 21221, NORTHERN -> 46877, PARK -> 23377, CENTRAL -> 41914, BAYVIEW -> 36111)
```

Caution! For large datasets, don't use countByKey.

Join Pair RDDs

This activity illustrates how joins work in Spark (Scala). There are two small datasets provided for this activity - J_AddCat.csv and J_AddDist.csv.

J_AddCat.csv - Category; Address			J_AddDist.csv - PdDistrict; Address		
1.	EMBEZZLEMENT	100_Block_of_JEFFERSON_ST	1.	INGLESIDE	100_Block_of_ROME_ST
2.	EMBEZZLEMENT	SUTTER_ST/MASON_ST	2.	SOUTHERN	0_Block_of_SOUTHPARK_AV
3.	BRIBERY	0_Block_of_SOUTHPARK_AV	3.	MISSION	1900_Block_of_MISSION_ST

4.	EMBEZZLEMENT	1500_Block_of_15TH_ST	4.	RICHMOND	1400_Block_of_CLEMENT_ST
5.	EMBEZZLEMENT	200_Block_of_BUSH_ST	5.	SOUTHERN	100_Block_of_BLUXOME_ST
6.	BRIBERY	1900_Block_of_MISSION_ST	6.	SOUTHERN	300_Block_of_BERRY_ST
7.	EMBEZZLEMENT	800_Block_of_MARKET_ST	7.	BAYVIEW	1400_Block_of_VANDYKE_AV
8.	EMBEZZLEMENT	2000_Block_of_MARKET_ST	8.	SOUTHERN	1100_Block_of_MISSION_ST
9.	BRIBERY	1400_Block_of_CLEMENT_ST	9.	TARAVAL	0_Block_of_CHUMASERO_DR
10.	EMBEZZLEMENT	1600_Block_of_FILLMORE_ST			
11.	EMBEZZLEMENT	1100_Block_of_SELBY_ST			
12.	BAD CHECKS	100_Block_of_BLUXOME_ST			

Based on the data above, answer the following questions:

1. Given these two datasets, you want to find the type of incident and district for each address. What is one way of doing this? (HINT: An operation on pairs or pairRDDs)

   What should the keys be for the two pairRDDs?

[You can use joins on pairs or pairRDD to get the information. The key for the pairRDD is the address.]

2. What is the size of the resulting dataset from a join? Why?

Note: Look at the datasets above and estimate what a join on the two would return if they were joined on the key - from #1.

Remember that a join is the same as an inner join and only keys that are present in both RDDs are output.

[A join is the same as an inner join and only keys that are present in both RDDs are output. If you compare the addresses in both the datasets, you find that there are five addresses in common and they are unique. Thus the resulting dataset will contain five elements. If there are multiple values for the same key, the resulting RDD will have an entry for every possible pair of values with that key from the two RDDs.]

3. If you did a right outer join on the two datasets with Address/Category being the source RDD, what would be the size of the resulting dataset? Why?

Note: Look at the datasets above and estimate what a join on the two would return if they were joined on the key - from #1.

Remember that a right outer join results in a pair RDD that has entries for each key in the other pair RDD.

[A right outer join results in a pair RDD that has entries for each key in the other pairRDD. If the source RDD contains data from J_AddCat.csv and the "other" RDD is represented by J_AddDist.csv, then since "other" RDD has 9 distinct addresses, the size of the result of a right outer join is 9.]

4. If you did a left outer join on the two datasets with Address/Category being the source

Note: Look at the datasets above and estimate what a join on the two would return if they were joined on the key - from #1.

Remember that a left outer join results in a pair RDD that has entries for each key in the source pair RDD.

RDD, what would be the size of the resulting dataset? Why?

[A left outer join results in a pair RDD that has entries for each key in the source pairRDD. If thesource RDD contains data from J_AddCat.csv and the "other" RDD is represented by J_AddDist.csv,then since "source" RDD has 13 distinct addresses, the size of the result of a left outer join is 13.]

5. Load each dataset into separate pairRDDs with “address” being the key.

Note: once you load the text file, split on the “,” and then apply the `map` transformation to create a pairRDD where address is the first element of the two-tuple.

```
#val catAdd = sc.textFile("J_AddCat.csv").map(x=>x.split(",")).map(x=>(x(1),x(0)))
#val distAdd = sc.textFile("J_AddDist.csv").map(x=>x.split(",")).map(x=>(x(1),x(0)))
```

6. List the incident category and district for those addresses that have both category and district information. Verify that the size estimated earlier is correct.

```
val catJdist = catAdd.join(distAdd)
catJdist.collect
catJdist.count
catJdist.take(10)
```

7. List the incident category and district for all addresses irrespective of whether each address has category and district information.

```
val catJdist1 = catAdd.leftOuterJoin(distAdd)
catJdist1.collect
catJdist.count
```

8. List the incident district and category for all addresses irrespective of whether each address has category and district information. Verify that the size estimated earlier is

correct.

```
val catJdist2 = catAdd.rightOuterJoin(distAdd)
catJdist2.collect
catJdist2.count
```

Sample O/P.

```
catJdist1: org.apache.spark.rdd.RDD[(String, (String, Option[String]))] = MapPartitionsRDD[45] at leftOuterJoin at <console>:27

scala> catJdist1.collect
res12: Array[(String, (String, Option[String]))] = Array((1600_Block_of_FILLMORE_ST,(EMBEZZLEMENT,None)), (1400_Block_of_CLEMENT_ST,(BRIBERY,Some(RICHMOND
))), (100_Block_of_BLUXOME_ST,(BAD CHECKS,Some(SOUTHERN))), (1100_Block_of_SELBY_ST,(EMBEZZLEMENT,None)), (1400_Block_of_VANDYKE_AV,(BRIBERY,Some(BAYVIEW)
)), (100_Block_of_JEFFERSON_ST,(EMBEZZLEMENT,None)), (SUTTER_ST/MASON_ST,(EMBEZZLEMENT,None)), (0_Block_of_SOUTHPARK_AV,(BRIBERY,Some(SOUTHERN))), (800_Bl
ock_of_MARKET_ST,(EMBEZZLEMENT,None)), (2000_Block_of_MARKET_ST,(EMBEZZLEMENT,None)), (1500_Block_of_15TH_ST,(EMBEZZLEMENT,None)), (200_Block_of_BUSH_ST,(
EMBEZZLEMENT,None)), (1900_Block_of_MISSION_ST,(BRIBERY,Some(MISSION))))

scala> catJdist.count
res13: Long = 5

scala>  val catJdist2 = catAdd.rightOuterJoin(distAdd)
catJdist2: org.apache.spark.rdd.RDD[(String, (Option[String], String))] = MapPartitionsRDD[48] at rightOuterJoin at <console>:27

scala> catJdist2.collect
res14: Array[(String, (Option[String], String))] = Array((300_Block_of_BERRY_ST,(None,SOUTHERN)), (1400_Block_of_CLEMENT_ST,(Some(BRIBERY),RICHMOND)), (10
0_Block_of_BLUXOME_ST,(Some(BAD CHECKS),SOUTHERN)), (1400_Block_of_VANDYKE_AV,(Some(BRIBERY),BAYVIEW)), (0_Block_of_SOUTHPARK_AV,(Some(BRIBERY),SOUTHERN))
, (0_Block_of_CHUMASERO_DR,(None,TARAVAL)), (1100_Block_of_MISSION_ST,(None,SOUTHERN)), (100_Block_of_ROME_ST,(None,INGLESIDE)), (1900_Block_of_MISSION_ST
,(Some(BRIBERY),MISSION)))

scala> catJdist2.count
res15: Long = 9

scala>
```

Complete Answer: (Skip the section till the partition section, its for reference only.)

Answer:

```
val catAdd =
sc.textFile("/opt/data/J_AddCat.csv").map(x=>x.split(",")).map(x=>(x(1),x(0)))
val distAdd =
sc.textFile("/opt/data/J_AddDist.csv").map(x=>x.split(",")).map(x=>(x(1),x(0)))
val catJdist=catAdd.join(distAdd)

catJdist.count
catJdist.take(10)
val catJdist1 = catAdd.leftOuterJoin(distAdd)
catJdist1.collect
catJdist.count
val catJdist2 = catAdd.rightOuterJoin(distAdd)
catJdist2.collect
catJdist2.count
```

Result

```
scala> val catAdd =
sc.textFile("/opt/data/J_AddCat.csv").map(x=>x.split(",")).map(x=>(x(1),x(0)))
catAdd: org.apache.spark.rdd.RDD[(String, String)] = MapPartitionsRDD[47] at map at
<console>:24

scala> val distAdd =
sc.textFile("/opt/data/J_AddDist.csv").map(x=>x.split(",")).map(x=>(x(1),x(0)))
```

```
distAdd: org.apache.spark.rdd.RDD[(String, String)] = MapPartitionsRDD[51] at map at
<console>:24

scala> val catJdist=catAdd.join(distAdd)
catJdist: org.apache.spark.rdd.RDD[(String, (String, String))] = MapPartitionsRDD[54] at
join at <console>:28



scala> catJdist.count
res18: Long = 5

scala> catJdist.take(10)
res19: Array[(String, (String, String))] =
Array((1400_Block_of_CLEMENT_ST,(BRIBERY,RICHMOND)),
(100_Block_of_BLUXOME_ST,(BAD CHECKS,SOUTHERN)),
(1400_Block_of_VANDYKE_AV,(BRIBERY,BAYVIEW)),
(0_Block_of_SOUTHPARK_AV,(BRIBERY,SOUTHERN)),
(1900_Block_of_MISSION_ST,(BRIBERY,MISSION)))

scala> val catJdist1 = catAdd.leftOuterJoin(distAdd)
catJdist1: org.apache.spark.rdd.RDD[(String, (String, Option[String]))] =
MapPartitionsRDD[60] at leftOuterJoin at <console>:28

scala> catJdist1.collect()
```

```
res20: Array[(String, (String, Option[String]))] =
Array((1600_Block_of_FILLMORE_ST,(EMBEZZLEMENT,None)),
(1400_Block_of_CLEMENT_ST,(BRIBERY,Some(RICHMOND))),
(100_Block_of_BLUXOME_ST,(BAD CHECKS,Some(SOUTHERN))),
(1100_Block_of_SELBY_ST,(EMBEZZLEMENT,None)),
(1400_Block_of_VANDYKE_AV,(BRIBERY,Some(BAYVIEW))),
(100_Block_of_JEFFERSON_ST,(EMBEZZLEMENT,None)),
(SUTTER_ST/MASON_ST,(EMBEZZLEMENT,None)),
(0_Block_of_SOUTHPARK_AV,(BRIBERY,Some(SOUTHERN))),
(800_Block_of_MARKET_ST,(EMBEZZLEMENT,None)),
(2000_Block_of_MARKET_ST,(EMBEZZLEMENT,None)),
(1500_Block_of_15TH_ST,(EMBEZZLEMENT,None)),
(200_Block_of_BUSH_ST,(EMBEZZLEMENT,None)),
(1900_Block_of_MISSION_ST,(BRIBERY,Some(MISSION))))

scala> catJdist1.collect()
res21: Array[(String, (String, Option[String]))] =
Array((1600_Block_of_FILLMORE_ST,(EMBEZZLEMENT,None)),
(1400_Block_of_CLEMENT_ST,(BRIBERY,Some(RICHMOND))),
(100_Block_of_BLUXOME_ST,(BAD CHECKS,Some(SOUTHERN))),
(1100_Block_of_SELBY_ST,(EMBEZZLEMENT,None)),
(1400_Block_of_VANDYKE_AV,(BRIBERY,Some(BAYVIEW))),
(100_Block_of_JEFFERSON_ST,(EMBEZZLEMENT,None)),
(SUTTER_ST/MASON_ST,(EMBEZZLEMENT,None)),
(0_Block_of_SOUTHPARK_AV,(BRIBERY,Some(SOUTHERN))),
```

```
(800_Block_of_MARKET_ST,(EMBEZZLEMENT,None)),
(2000_Block_of_MARKET_ST,(EMBEZZLEMENT,None)),
(1500_Block_of_15TH_ST,(EMBEZZLEMENT,None)),
(200_Block_of_BUSH_ST,(EMBEZZLEMENT,None)),
(1900_Block_of_MISSION_ST,(BRIBERY,Some(MISSION))))

scala> catJdist.count
res22: Long = 5
scala> val catJdist2 = catAdd.rightOuterJoin(distAdd)
catJdist2: org.apache.spark.rdd.RDD[(String, (Option[String], String))] =
MapPartitionsRDD[63] at rightOuterJoin at <console>:28

scala> catJdist2.collect
res23: Array[(String, (Option[String], String))] =
Array((300_Block_of_BERRY_ST,(None,SOUTHERN)),
(1400_Block_of_CLEMENT_ST,(Some(BRIBERY),RICHMOND)),
(100_Block_of_BLUXOME_ST,(Some(BAD CHECKS),SOUTHERN)),
(1400_Block_of_VANDYKE_AV,(Some(BRIBERY),BAYVIEW)),
(0_Block_of_SOUTHPARK_AV,(Some(BRIBERY),SOUTHERN)),
(0_Block_of_CHUMASERO_DR,(None,TARAVAL)),
(1100_Block_of_MISSION_ST,(None,SOUTHERN)),
(100_Block_of_ROME_ST,(None,INGLESIDE)),
(1900_Block_of_MISSION_ST,(Some(BRIBERY),MISSION)))

scala> catJdist2.count
```

res24: Long = 9

Explore Partitioning

In this activity we see how to determine the number of partitions, the type of partitioner and how to specify partitions in a transformation.

Objective

- Explore partitioning in RDDs

Explore partitioning in RDDs

Note To find partition size: rdd.partitions.size (Scala); rdd.getNumPartitions() (in Python)

To determine the partitioner: rdd.partitioner (Scala);

1. How many partitions are there in the sfpdRDD?
   sfpdRDD.partitions.size
2. How do you find the type of partitioner for sfpdRDD?
   sfpdRDD.partitioner
3. Create a pair RDD.

```
val incByDists = sfpdRDD.map(incident=>(incident(6),1)).reduceByKey((x,y)=>x+y)
```

How many partitions does incByDists have?

incByDists.partitions.size

What type of partitioner does incByDists have?

```
sfpdRDD.partitioner
```

Q: Why does incByDists have that partitioner?

A: reduceByKey automatically uses the Hash Partitioner

4. Add a map

```
val inc_map = incByDists.map(x=>(x._2,x._1))
```

How many partitions does inc_map have?

```
inc_map.partitions.size
```

What type of partitioner does incByDists have? incByDists.partitioner

Q: Why does inc_map not have the same partitioner as its parent?

A: Transformations such as map() cause the new RDD to forget the parent's partitioning information because a map() can modify the key of each record.

5. Add a sortByKey

```
val inc_sort = inc_map.sortByKey(false)
```

What type of partitioner does inc_sort have?

```
inc_sort.partitioner
```

Q: Why does inc_sort have this type of partitioner?

A: This is because sortByKey will automatically result in a range partitioned RDD..

6. Add groupByKey

```
val inc_group = sfpdRDD.map(incident=>(incident(6),1)).groupByKey()
```

What type of partitioner does inc_group have? inc_group.partitioner

Q: Why does inc_group have this type of partitioner?

A: This is because groupByKey will automatically result in a hash partitioned RDD..

7. Create two pairRDD

```
val catAdd = sc.textFile("J_AddCat.csv").map(x=>x.split(",")).map(x=>(x(1),x(0)))
```

```
val distAdd =
sc.textFile("J_AddDist.csv").map(x=>x.split(",")).map(x=>(x(1)
,x(0)))
```

8. You can specify the number of partitions when you use the join operation.

```
val catJdist = catAdd.join(distAdd,8)
```

How many partitions does the joined RDD have? catJdist.partitions.size

What type of partitioner does catJdist have? catJdist.partitioner

Answer:

1. **sfpdRDD.partitions.size**
2. **sfpdRDD.partitioner**
3. **incByDists.partitions.size;**
   - **incByDists.partitioner**
4. **inc_map.partitions.size;**
   - **inc_map.partitioner**
5. **inc_sort.partitioner**
6. **inc_group.partitioner**
7. **incByDists.partitions.size**

8 **catJdist.partitions.size**

catJdist.partitioner

Result

```
scala>      val catAdd =
sc.textFile("/opt/data/J_AddCat.csv").map(x=>x.split(",")).map(x=>(x(1),x(0)))
catAdd: org.apache.spark.rdd.RDD[(String, String)] = MapPartitionsRDD[3] at map at
<console>:24

scala> val distAdd =
sc.textFile("/opt/data/J_AddDist.csv").map(x=>x.split(",")).map(x=>(x(1),x(0)))
distAdd: org.apache.spark.rdd.RDD[(String, String)] = MapPartitionsRDD[7] at map at
<console>:24

scala> val catJdist=catAdd.join(distAdd)
catJdist: org.apache.spark.rdd.RDD[(String, (String, String))] = MapPartitionsRDD[10] at
join at <console>:28

scala> val sfpdRDD = sc.textFile("/opt/data/sfpd.csv").map(_.split(","))
sfpdRDD: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[13] at map at
<console>:24

scala> sfpdRDD.partitions.size
res2: Int = 2

scala> sfpdRDD.partitioner
res3: Option[org.apache.spark.Partitioner] = None
```

```
scala> val incByDists = sfpdRDD.map(incident=>(incident(6),1)).reduceByKey((x,y)
    | =>x+y)
incByDists: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[15] at reduceByKey at
<console>:26

scala> incByDists.partitions.size; incByDists.partitioner
res4: Option[org.apache.spark.Partitioner] = Some(org.apache.spark.HashPartitioner@2)

scala> incByDists.partitions.size;
res5: Int = 2
scala>  incByDists.partitioner
res6: Option[org.apache.spark.Partitioner] = Some(org.apache.spark.HashPartitioner@2)

scala> val inc_map = incByDists.map(x=>(x._2,x._1))
inc_map: org.apache.spark.rdd.RDD[(Int, String)] = MapPartitionsRDD[16] at map at
<console>:28

scala> inc_map.partitions.size;
res7: Int = 2

scala> inc_map.partitioner
res8: Option[org.apache.spark.Partitioner] = None

scala> val inc_sort = inc_map.sortByKey(false)
```

```
inc_sort: org.apache.spark.rdd.RDD[(Int, String)] = ShuffledRDD[19] at sortByKey at
<console>:30

scala> inc_sort.partitioner
res9: Option[org.apache.spark.Partitioner] =
Some(org.apache.spark.RangePartitioner@fb7ef)

scala> val inc_group = sfpdRDD.map(incident=>(incident(PdDistrict),1)).groupByKey()
<console>:26: error: not found: value PdDistrict
     val inc_group = sfpdRDD.map(incident=>(incident(PdDistrict),1)).groupByKey()
                                            ^

scala> val inc_group = sfpdRDD.map(incident=>(incident(6),1)).groupByKey()
inc_group: org.apache.spark.rdd.RDD[(String, Iterable[Int])] = ShuffledRDD[21] at
groupByKey at <console>:26

scala> inc_group.partitioner
res10: Option[org.apache.spark.Partitioner] = Some(org.apache.spark.HashPartitioner@2)

scala> val catAdd =
sc.textFile("/opt/data/J_AddCat.csv").map(x=>x.split(",")).map(x=>(x(1),x(0)))
catAdd: org.apache.spark.rdd.RDD[(String, String)] = MapPartitionsRDD[25] at map at
<console>:24
```

```
scala> val distAdd =
sc.textFile("/opt/data/J_AddDist.csv").map(x=>x.split(",")).map(x=>(x(1),x(0)))
distAdd: org.apache.spark.rdd.RDD[(String, String)] = MapPartitionsRDD[29] at map at
<console>:24

scala> val catJdist = catAdd.join(distAdd,8)
catJdist: org.apache.spark.rdd.RDD[(String, (String, String))] = MapPartitionsRDD[32] at
join at <console>:28

scala> catJdist.partitions.size
res11: Int = 8

scala> catJdist.partitioner
res14: Option[org.apache.spark.Partitioner] = Some(org.apache.spark.HashPartitioner@8)
```

Word Count Using Pair.

```
val counts = sc.textFile("J_AddCat.csv").flatMap(line => line.split(',')).map(word =>
(word,1))
val result = counts.reduceByKey((v1,v2) => v1+v2)
result.collect()
```

```
scala> val counts = sc.textFile("J_AddCat.csv").flatMap(line => line.split(',')).map(word => (word,1))
counts: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[76] at map at <console>:24

scala> val result = counts.reduceByKey((v1,v2) => v1+v2)
result: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[77] at reduceByKey at <console>:25

scala> result.take(12)
res32: Array[(String, Int)] = Array((1600_Block_of_FILLMORE_ST,1), (1400_Block_of_CLEMENT_ST,1), (EMBEZZLEMENT,8), (100_Block_of_BLUXOME_ST,1), (1100_Bloc
k_of_SELBY_ST,1), (1400_Block_of_VANDYKE_AV,1), (BAD CHECKS,1), (100_Block_of_JEFFERSON_ST,1), (SUTTER_ST/MASON_ST,1), (0_Block_of_SOUTHPARK_AV,1), (BRIBE
RY,4), (800_Block_of_MARKET_ST,1))

scala>
```

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

11. Zeppelin Installation

http://zeppelin.apache.org/download.html

zeppelin-0.9.0-bin-all.tgz

You don't require spark installation separately.

#tar -xvf zeppelin-0.9* -C /opt

set the JAVA_HOME [vi ~/.bashrc]
export JAVA_HOME=/opt/jdk
export PATH=:$JAVA_HOME/bin:$PATH:$SCALA_HOME/bin

#cp zeppelin-site.xml.template zeppelin-site.xml

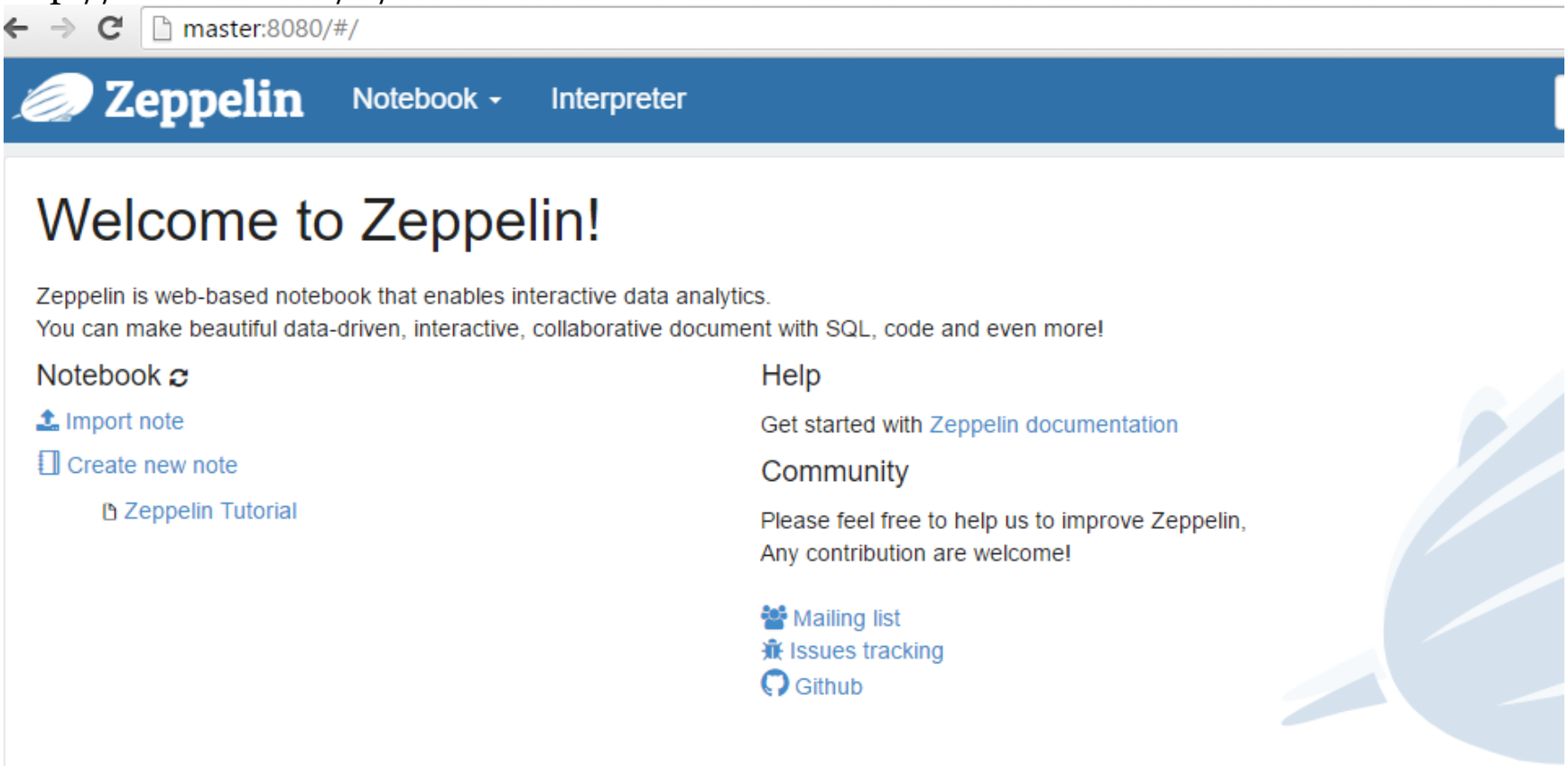
Mention the hostname of the node.

```
<property>
  <name>zeppelin.server.addr</name>
  <value>hp.tos.com</value>
  <description>Server address</description>
</property>

<property>
  <name>zeppelin.server.port</name>
  <value>8081</value>
  <description>Server port.</description>
</property>
```

Start Zeppelin
cd /opt/zeppelin
bin/zeppelin-daemon.sh start
jps
Note: You don't require to start the spark cluster before starting Zeppelin

```
[zeppelin@hp zeppelin-0.7.3]$ pwd
/apps/zeppelin-0.7.3
[zeppelin@hp zeppelin-0.7.3]$ ls
bin   interpreter  LICENSE   local-repo  notebook  README.md  webapps
conf  lib          licenses  logs        NOTICE    run        zeppelin-web-0.7.3.war
[zeppelin@hp zeppelin-0.7.3]$ bin/zeppelin-daemon.sh start
Zeppelin start                                             [  OK  ]
[zeppelin@hp zeppelin-0.7.3]$ jps
2887 ZeppelinServer
2911 Jps
```

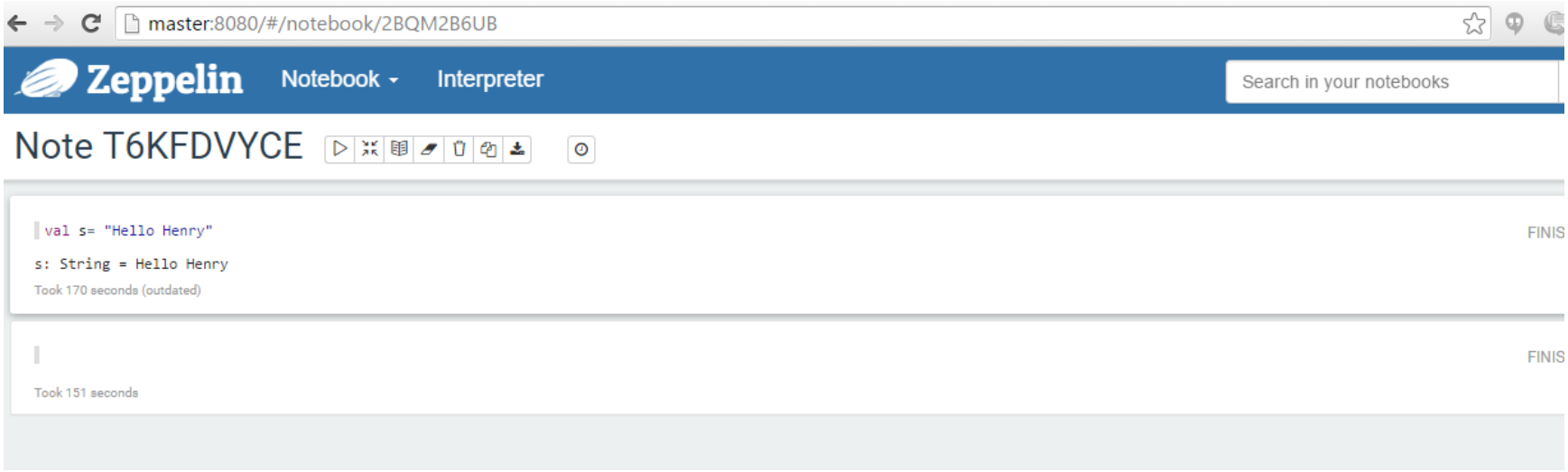
http://master:8081/#/



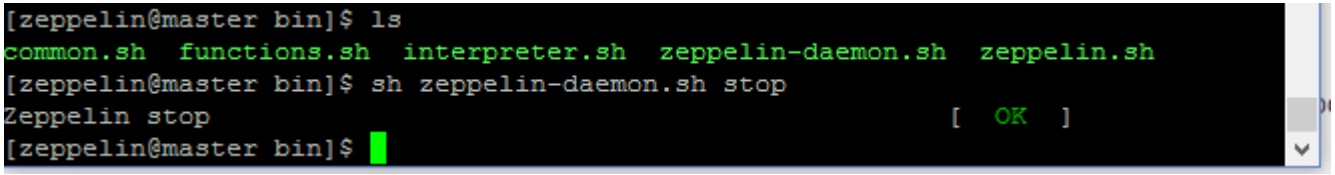
Notebook → Create new Note

Enter Note Name – Learning Spark

Interpreter – Spark.



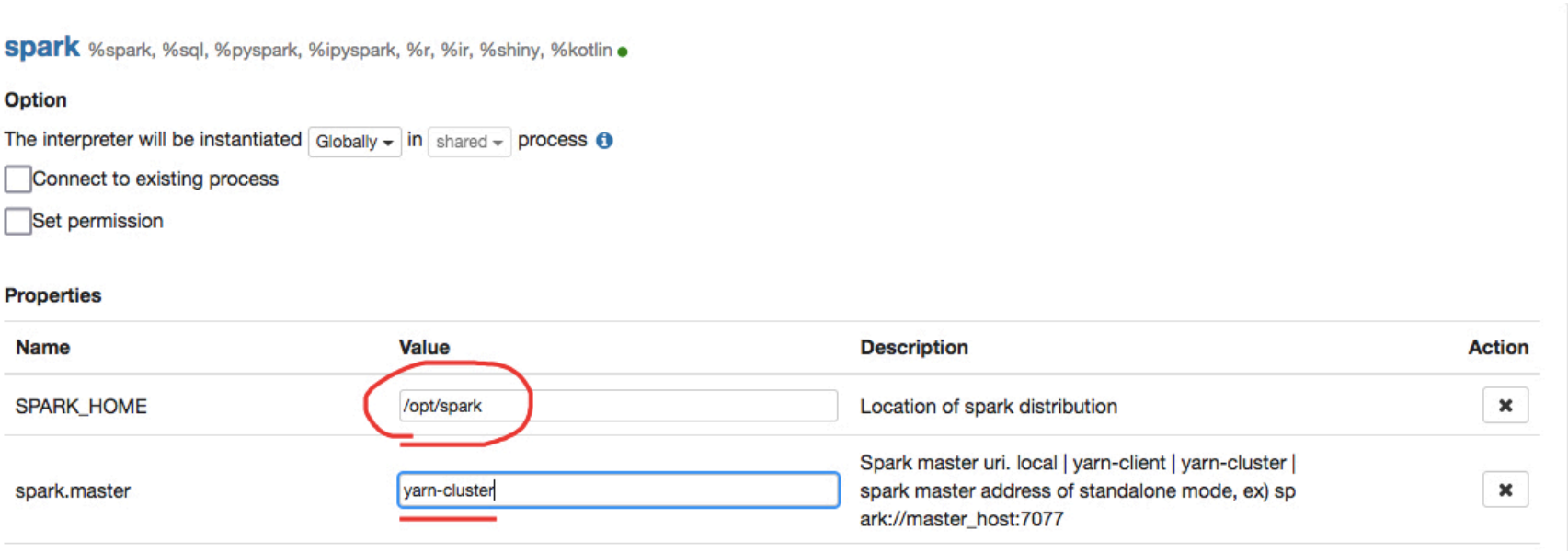
Stop Zeppelin
bin/zeppelin-daemon.sh stop



Connect to Hadoop Cluster

1. Specify `SPARK_HOME` in interpreter setting. If you don't specify `SPARK_HOME`, Zeppelin will use the embedded spark which can only run in local mode. And some advanced features may not work in this embedded spark.
2. Specify `master` for spark execution mode.

Yarn or `yarn-cluster`



Click Save.

```
cp /home/zeppelin/zeppelin/conf/zeppelin-env.sh.template /home/zeppelin/zeppelin/conf/zeppelin-env.sh
```

Set the following properties

```
export JAVA_HOME="/usr/java/jdk1.7.0_79"
export HADOOP_CONF_DIR="/etc/hadoop/conf"
```

Need to be verify

Let us connect Zeppelin with the existing Spark Cluster. You can start Spark in a single standalone cluster or Use any existing Spark Cluster.

In this lab let us use the single standalone spark cluster,

Let us set the Spark Metadata to be stored in derby Network so that more than one user can connect using spark shell.

Unzip the derby database and start as below: You can use zeppelin user account to start the derby.

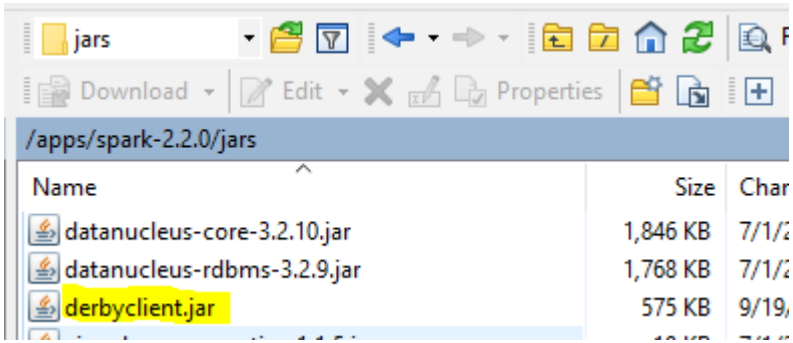
```
#tar -xzf db-derby-10.14.1.0-bin.tar.gz -C /apps
#mkdir db-derby-10.4.1.3-bin/data
#nohup /apps/db-derby-10.14.1.0-bin/bin/startNetworkServer -h 0.0.0.0 &
```

After the derby server, you need to modify the spark to refer the above database server. For this create a file hive-site.xml in the SPARK_HOME/conf folder and paste the following content,

```
<configuration>
        <property>
          <name>javax.jdo.option.ConnectionURL</name>
          <value>jdbc:derby://10.10.20.27:1527/metastore_db;create=true</value>
        </property>
        <property>
          <name>javax.jdo.option.ConnectionDriverName</name>
          <value>org.apache.derby.jdbc.ClientDriver</value>
```

```
    </property>
</configuration>
```

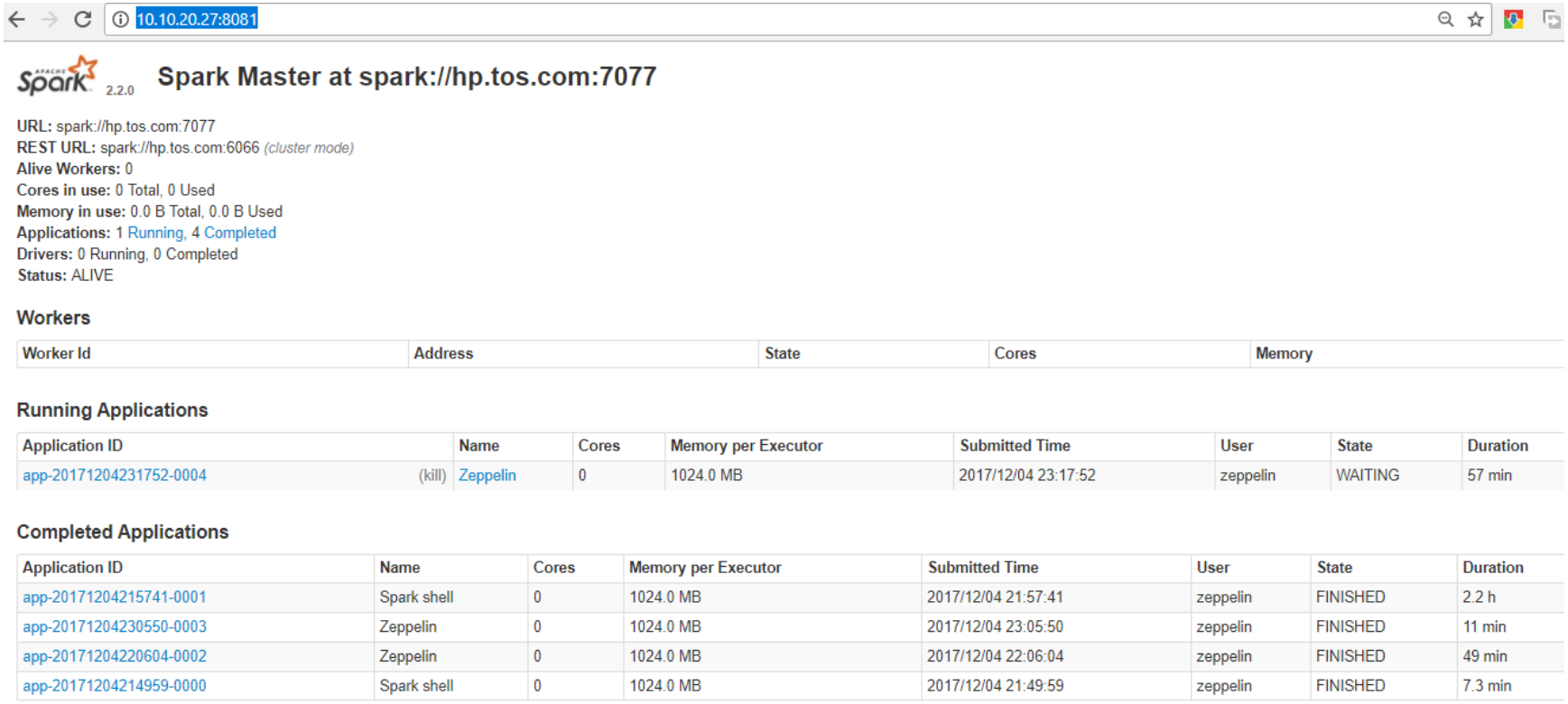
Remove the derby jar from the Spark installation jar folder and replace with the derby client jar.



After that start the spark standalone cluster:

sbin/start-master.sh --webui-port 8081

At the end of the above step you should be able to access the Spark UI
http://10.10.20.27:8081/



10.10.20.27:8081

Spark 2.2.0

Spark Master at spark://hp.tos.com:7077

URL: spark://hp.tos.com:7077
REST URL: spark://hp.tos.com:6066 *(cluster mode)*
Alive Workers: 0
Cores in use: 0 Total, 0 Used
Memory in use: 0.0 B Total, 0.0 B Used
Applications: 1 Running, 4 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory

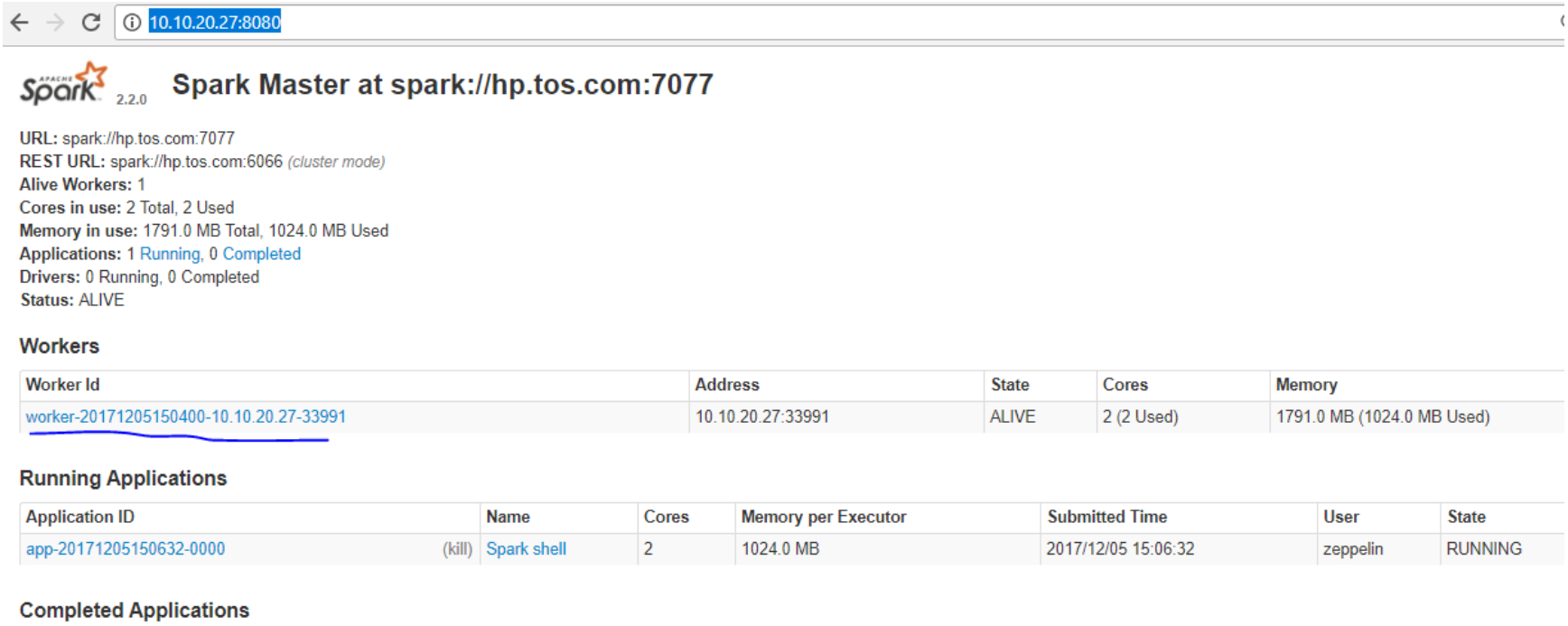
Running Applications

Application ID		Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20171204231752-0004	(kill)	Zeppelin	0	1024.0 MB	2017/12/04 23:17:52	zeppelin	WAITING	57 min

Completed Applications

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20171204215741-0001	Spark shell	0	1024.0 MB	2017/12/04 21:57:41	zeppelin	FINISHED	2.2 h
app-20171204230550-0003	Zeppelin	0	1024.0 MB	2017/12/04 23:05:50	zeppelin	FINISHED	11 min
app-20171204220604-0002	Zeppelin	0	1024.0 MB	2017/12/04 22:06:04	zeppelin	FINISHED	49 min
app-20171204214959-0000	Spark shell	0	1024.0 MB	2017/12/04 21:49:59	zeppelin	FINISHED	7.3 min

Now, we have configured SPARK standalone cluster. We need to have atleast one worker node as shown below:
Add hp.tos.com in the slave file and start the slave with the following command
sh start-slave.sh spark://hp.tos.com:7077
http://10.10.20.27:8080/



Let us configure zeppelin to connect to it.

Stop the zeppelin if not done before. Use zeppelin user ID to execute the following command

#bin/zeppelin-daemon.sh stop

open the **zeppelin-env.sh** file present in **$zeppelin_HOME/conf** directory and provide the below specified configurations.

export MASTER=spark://hp.tos.com:7077
export SPARK_HOME=/apps/spark-2.2.0

Save the above file.

Let us activate admin user for logon to zeppelin web UI.

#cd /apps/zeppelin-0.7.3/conf

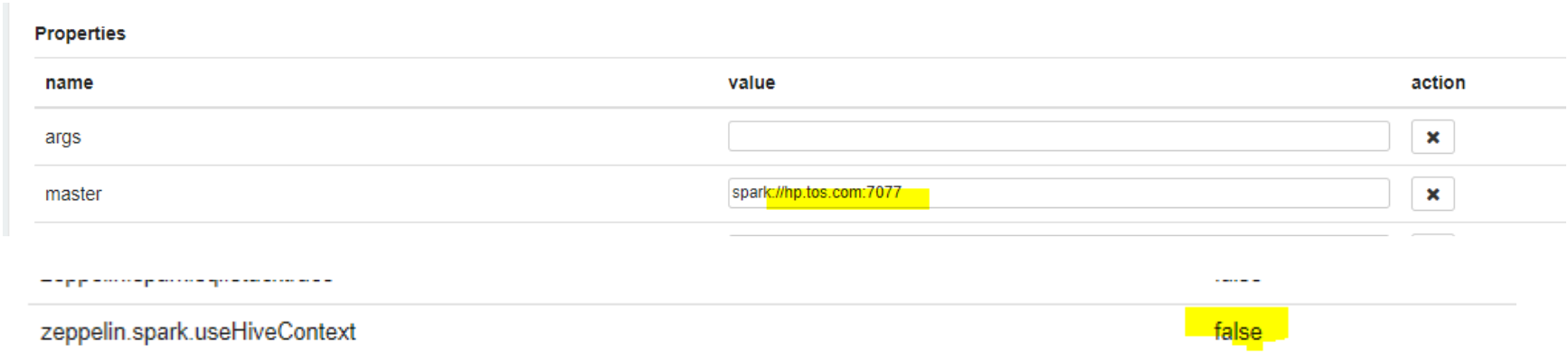
#cp shiro.ini.template shiro.ini
update the password of admin as shown below in the above ini file.
admin = life213

start the zeppelin using zeppelin user ID.
#bin/zeppelin-daemon.sh start

Now open your Zeppelin dashboard and go to the list of interprets and search for Spark interpreter.

Ensure that you modify the below parameters: You need to logon using admin/life213 credentials.

Interpreter → Spark → Edit



Save and then accept the prompt by clicking Ok button.

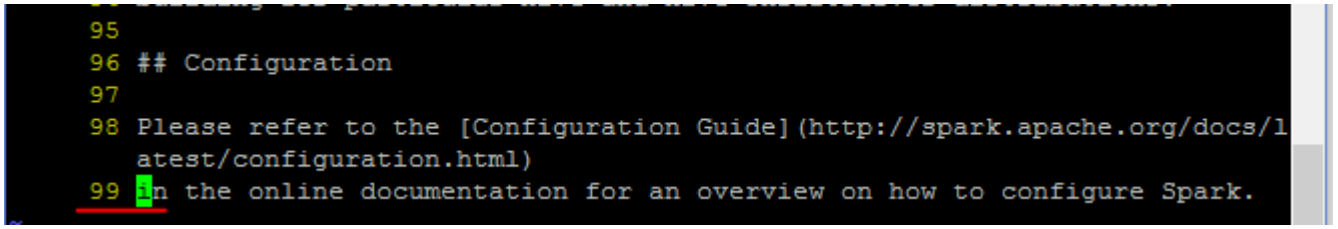
Create a New Notebook [Notebook → Create New Note → Enter Hello Spark as the name]

Enter sc and the click Run which is on the right corner. If everything is ok, you should be able to see as the above screenshot. Else review the log file in zeppelin installation folder.

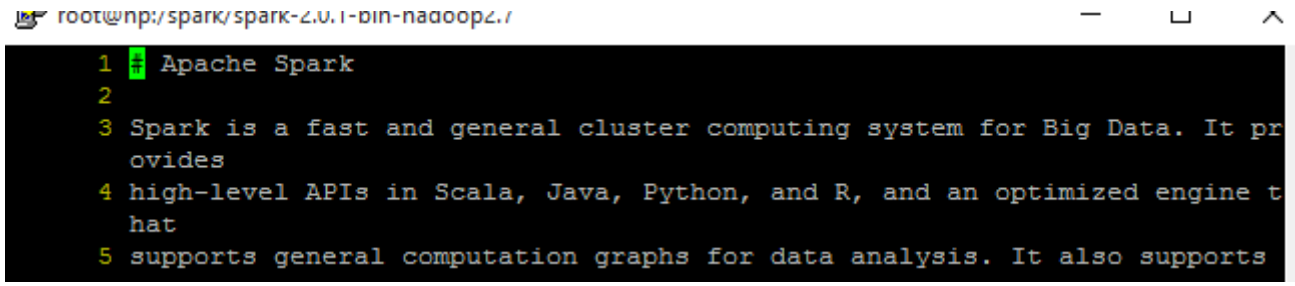
12. Spark Scala Example – Using Zepplein

Familiarization with Zepplein/Scala

```
var lines =  sc.textFile("/opt/zeppelin/README.md");
lines.count()
lines.first()
```



Vi , set nu. $ goto the end of file



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

13. Spark SQL – 30 Minutes

Copy the person.txt file to /software folder.

```
Henry,42
Rajnita,40
Henderson,14
Tiraj,5
```

Open a terminal from /software folder and enter the following command.

```
#spark-shell
// sc is an existing SparkContext.
import sqlContext._
import sqlContext.implicits._

// Define the schema using a case class.
case class Person(name: String, age: Int)
```

```
scala> val sqlContext = new org.apache.spark.sql.SQLContext(sc)
sqlContext: org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@5c
1817

scala> import sqlContext._
import sqlContext._

scala> import sqlContext.implicits._
import sqlContext.implicits._

scala> case class Person(name: String, age: Int)
defined class Person
```

```
// Create an RDD of Person objects and register it as a table.
val people = sc.textFile("person.txt").map(_.split(",")).map(p => Person(p(0),
p(1).trim.toInt)).toDF()
people.createOrReplaceTempView("people")


// SQL statements can be run by using the sql methods provided by sqlContext.
val kids = spark.sql("SELECT name FROM people WHERE age >= 1 AND age <= 9")
```

```
scala> val teenagers = sqlContext.sql("SELECT name FROM people WHERE age >= 13 A
ND age <= 19")
teenagers: org.apache.spark.sql.DataFrame = [name: string]
```

```
// The results of SQL queries are DataFrames and support all the normal RDD operations.
kids.map(t => "Name: " + t(0)).collect().foreach(println)
```

```
scala> val kids = spark.sql("SELECT name FROM people WHERE age >= 1 AND age <= 9")
kids: org.apache.spark.sql.DataFrame = [name: string]

scala> kids.map(t => "Name: " + t(0)).collect().foreach(println)
Name: Tiraj

scala> kids.map(t => "Name: " + t(0)).collect().foreach(println)
```

Create a temporary view:

people.createTempView("people1")

Display the output.

spark.sql("select * from people1").show

```
scala> people.createTempView("people1")

scala> spark.sql("select * from people1")
res9: org.apache.spark.sql.DataFrame = [name: string, age: int]

scala> spark.sql("select * from people1").show
+---------+---+
|     name|age|
+---------+---+
|    Henry| 42|
|  Rajnita| 40|
|Henderson| 14|
|    Tiraj|  5|
+---------+---+
```

Show the Tables or View List.

spark.catalog.listTables.show

```
scala> spark.catalog.listTables.show
20/11/09 09:17:34 WARN HiveConf: HiveConf of name hive.stats.jdbc.timeout does not exist
20/11/09 09:17:34 WARN HiveConf: HiveConf of name hive.stats.retries.wait does not exist
20/11/09 09:17:44 WARN ObjectStore: Version information not found in metastore. hive.metastore.schema.veri
fication is not enabled so recording the schema version 2.3.0
20/11/09 09:17:44 WARN ObjectStore: setMetaStoreSchemaVersion called but recording version is disabled: ve
rsion = 2.3.0, comment = Set by MetaStore UNKNOWN@172.18.0.2
20/11/09 09:17:44 WARN ObjectStore: Failed to get database default, returning NoSuchObjectException
20/11/09 09:17:45 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
+-------+--------+-----------+---------+-----------+
|   name|database|description|tableType|isTemporary|
+-------+--------+-----------+---------+-----------+
| people|    null|       null|TEMPORARY|       true|
|people1|    null|       null|TEMPORARY|       true|
+-------+--------+-----------+---------+-----------+
```

Using DataFrame:

/software/people.json

```
{"name":"Michael"}
{"name":"Andy",
"age":30}
{"name":"Justin",
"age":19}
```

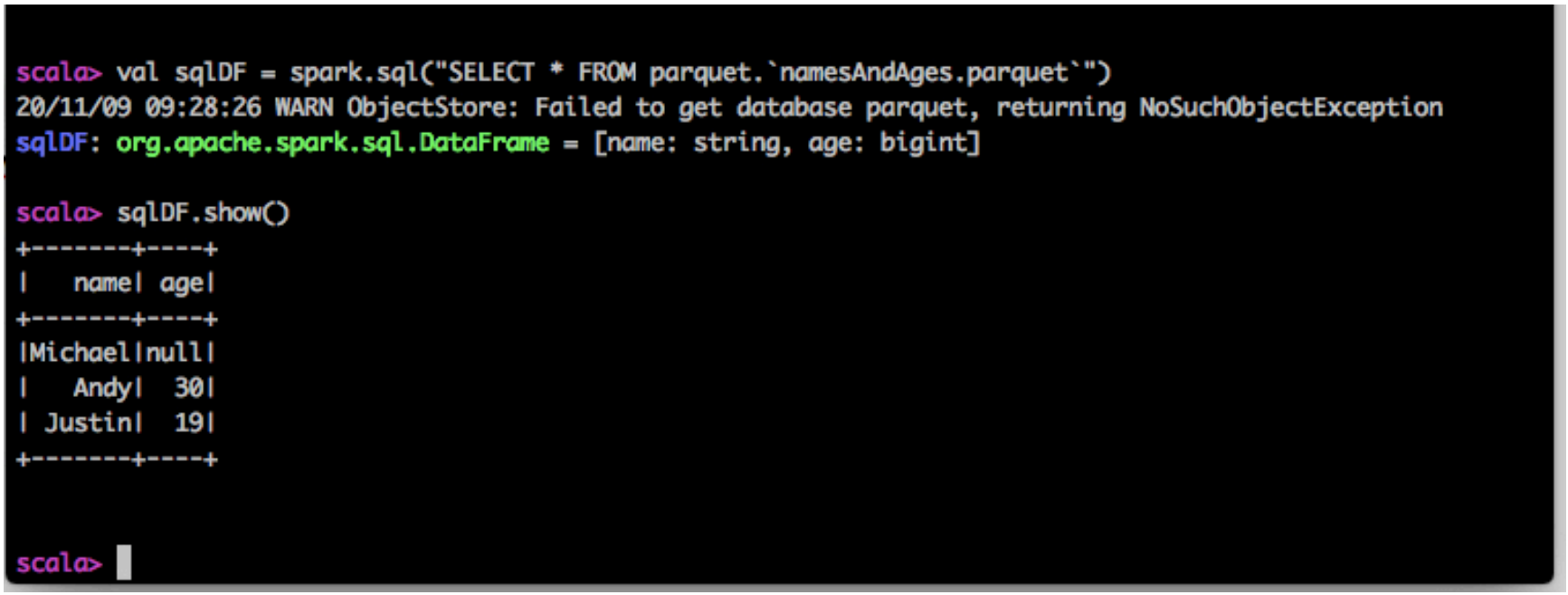
```
val peopleDF = spark.read.format("json").load("people.json")

peopleDF.select("name", "age").write.format("parquet").save("namesAndAges.parquet")

peopleDF.select("name","age").show
```

```
scala> val peopleDF = spark.read.format("json").load("people.json")
peopleDF: org.apache.spark.sql.DataFrame = [age: bigint, name: string]

scala> peopleDF.select("name", "age").write.format("parquet").save("namesAndAges.parquet")

scala> peopleDF.select("name","age").show
+-------+----+
|   name| age|
+-------+----+
|Michael|null|
|   Andy|  30|
| Justin|  19|
+-------+----+


scala>
```

Query file directly with SQL.

```
val sqlDF = spark.sql("SELECT * FROM parquet.`namesAndAges.parquet`")
```

sqlDF.show()

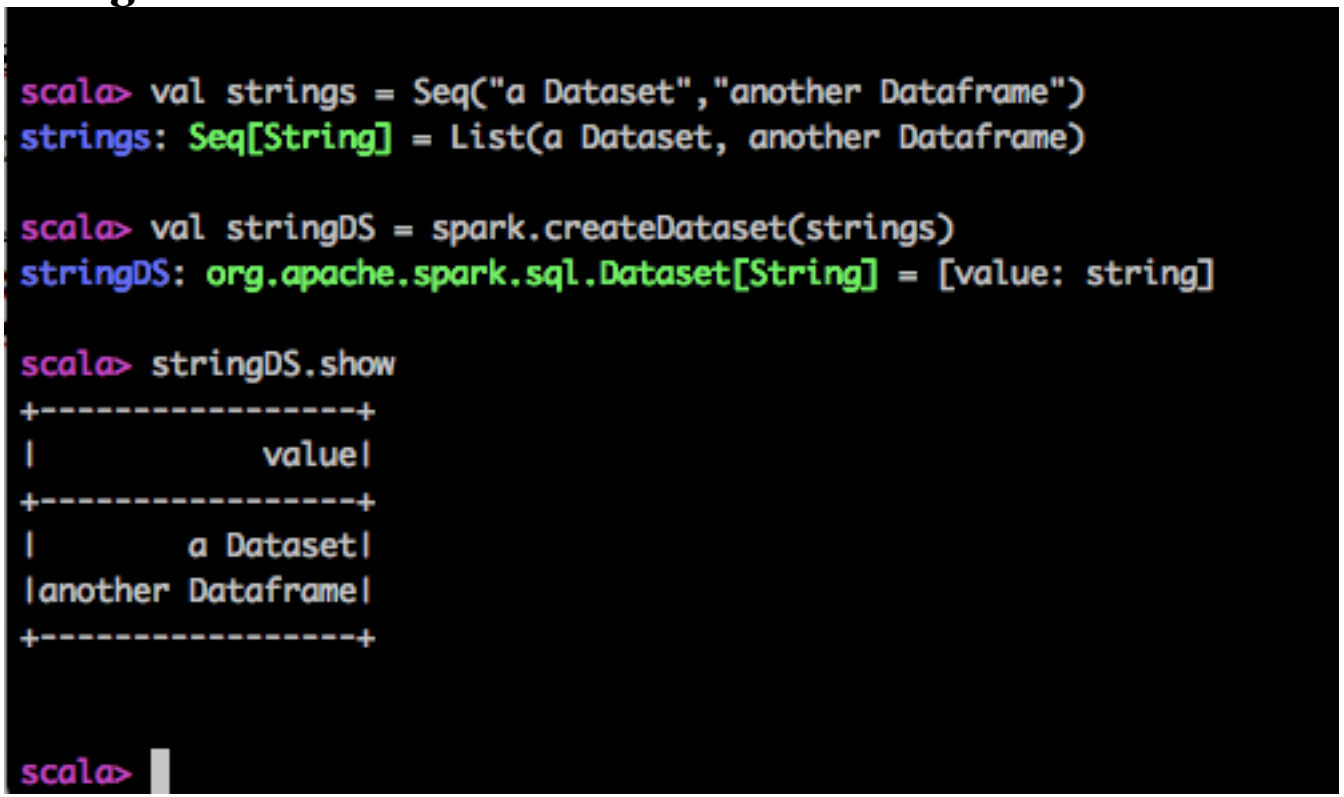


---------- Lab Ends Here ----------

14. Using DataSet – 45 Minutes

Initializing Dataset from Memory.

val strings = Seq("a Dataset","another Dataframe")
val stringDS = spark.createDataset(strings)
stringDS.show



Use case class and Encoders

```
case class Name(firstName: String, lastName: String)
val names = Seq(Name("Fred","Flintstone"),Name("Barney","Rubble"))
names.foreach(name => println(name.firstName))

import spark.implicits._ // required if not running in shell
val namesDS = spark.createDataset(names)
namesDS.show
```

```
scala> case class Name(firstName: String, lastName: String)
defined class Name

scala> val names = Seq(Name("Fred","Flintstone"),Name("Barney","Rubble"))
names: Seq[Name] = List(Name(Fred,Flintstone), Name(Barney,Rubble))

scala> import spark.implicits._ // required if not running in shell
import spark.implicits._

scala> val namesDS = spark.createDataset(names)
namesDS: org.apache.spark.sql.Dataset[Name] = [firstName: string, lastName: string]

scala> namesDS.show
+---------+----------+
|firstName|  lastName|
+---------+----------+
|     Fred|Flintstone|
|   Barney|    Rubble|
+---------+----------+
```

Creating a Dataset from a DataFrame
/software/names.json

{"firstName":"Grace","lastName":"Hopper"}
{"firstName":"Alan","lastName":"Turing"}

{"firstName":"Ada","lastName":"Lovelace"}
{"firstName":"Charles","lastName":"Babbage"}

Creating a Dataset from a DataFrame

val namesDF = spark.read.json("names.json")
namesDF.show

```
scala> val namesDF = spark.read.json("names.json")
namesDF: org.apache.spark.sql.DataFrame = [firstName: string, lastName: string]

scala> namesDF.show
+---------+--------+
|firstName|lastName|
+---------+--------+
|    Grace|  Hopper|
|     Alan|  Turing|
|      Ada|Lovelace|
|  Charles| Babbage|
+---------+--------+
```

case class Name(firstName: String, lastName: String)
val namesDS = namesDF.as[Name]

namesDS.show

```
scala> case class Name(firstName: String, lastName: String)
defined class Name

scala> val namesDS = namesDF.as[Name]
namesDS: org.apache.spark.sql.Dataset[Name] = [firstName: string, lastName: string]

scala> namesDS.show
+---------+--------+
|firstName|lastName|
+---------+--------+
|    Grace|  Hopper|
|     Alan|  Turing|
|      Ada|Lovelace|
|  Charles| Babbage|
+---------+--------+


scala> namesDS.printSchema
   def printSchema(level: Int): Unit   def printSchema(): Unit

scala> namesDS.printSchema
root
 |-- firstName: string (nullable = true)
 |-- lastName: string (nullable = true)
```

Creating Datasets from RDDs

/software/postalcode.txt (Tab space should be present between the attributes)

```
00210		43.005895	71.013202
01014		42.170731-72.604842
01062		42.324232	-72.67915
```

```
case class PcodeLatLon(pcode: String,latlon: Tuple2[Double,Double])

val pLatLonRDD = sc.textFile("postalcode.txt").map(line => line.split('\t')).map(fields =>
(PcodeLatLon(fields(0),
(fields(1).toFloat,fields(2).toFloat))))

val pLatLonDS = spark.createDataset(pLatLonRDD)

pLatLonDS.printSchema

pLatLonDS.show()
```

```
scala> val pLatLonRDD = sc.textFile("postalcode.txt").map(line => line.split('\t')).map(fields =>  (PcodeLatLon
(fields(0),(fields(1).toFloat,fields(2).toFloat))))
pLatLonRDD: org.apache.spark.rdd.RDD[PcodeLatLon] = MapPartitionsRDD[27] at map at <console>:26

scala> val pLatLonDS = spark.createDataset(pLatLonRDD)
pLatLonDS: org.apache.spark.sql.Dataset[PcodeLatLon] = [pcode: string, latlon: struct<_1: double, _2: double>]

scala> pLatLonDS.printSchema
root
 |-- pcode: string (nullable = true)
 |-- latlon: struct (nullable = true)
 |    |-- _1: double (nullable = false)
 |    |-- _2: double (nullable = false)


scala> pLatLonDS.show
+-----+------------------+
|pcode|            latlon|
+-----+------------------+
|00210|[43.0058937072753...|
|01014|[42.1707305908203...|
|01062|[42.3242301940918...|
+-----+------------------+
```

Typed and Untyped Transformations

case class Person(pcode:String, lastName:String,firstName:String, age:Int)

val people = Seq(Person("02134","Hopper","Grace",48))
 val peopleDS = spark.createDataset(people)

val sortedDS = peopleDS.sort("age")

```
val firstLastDF = peopleDS.select("firstName","lastName")

val combineDF = peopleDS.sort("lastName").where("age >
40").select("firstName","lastName")

combineDF.show
```

```
scala> case class Person(pcode:String, lastName:String,firstName:String, age:Int)
defined class Person

scala> val people = Seq(Person("02134","Hopper","Grace",48))
people: Seq[Person] = List(Person(02134,Hopper,Grace,48))

scala>  val peopleDS = spark.createDataset(people)
peopleDS: org.apache.spark.sql.Dataset[Person] = [pcode: string, lastName: string ... 2 more fields]

scala> val sortedDS = peopleDS.sort("age")
sortedDS: org.apache.spark.sql.Dataset[Person] = [pcode: string, lastName: string ... 2 more fields]

scala> val firstLastDF = peopleDS.select("firstName","lastName")
firstLastDF: org.apache.spark.sql.DataFrame = [firstName: string, lastName: string]

scala> val combineDF = peopleDS.sort("lastName").where("age > 40").select("firstName","lastName")
combineDF: org.apache.spark.sql.DataFrame = [firstName: string, lastName: string]

scala> combineDF.show
+---------+--------+
|firstName|lastName|
+---------+--------+
|    Grace|  Hopper|
+---------+--------+
```

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

15. Running a Spark Application – 45 Minutes

Using CLI – SBT

Install Scala Build Tools. You can refer the following documents for your respective OS.

https://www.scala-sbt.org/download.html

https://www.scala-sbt.org/1.x/docs/Installing-sbt-on-Linux.html

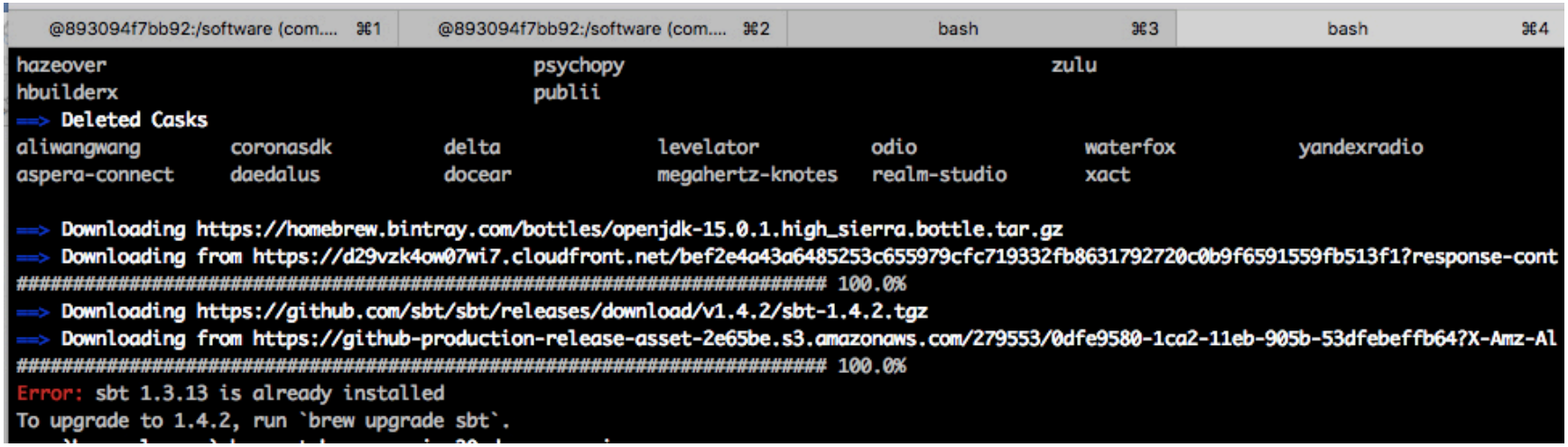
Mac Os:

brew install sbt

Linux:

```
curl https://bintray.com/sbt/rpm/rpm |  tee /etc/yum.repos.d/bintray-sbt-rpm.repo
```

yum install sbt



Create a directory to store the code and its build command.

#mkdir spark-sbt
cd spark-sbt
touch build.sbt

Update the build.sbt with the following instructions:

```
name := "Simple Project"
version := "1.0"
scalaVersion := "2.12.10"
libraryDependencies += "org.apache.spark" %% "spark-sql" % "3.0.1"
```