Table of Contents

Spark Resilient Distributed Datasets (RDDs)

The following is a helper lab for RDDs. As we have learned, Apache Spark is engineered to optimize in-memory computing in order to process massive amounts of data. It does so by scaling thousands of commodity servers horizontally. Working between all of these clustered servers requires coordination. When coordinated, a cluster of servers can make quick work out of large datasets using the power of distributed and parallel computing.

Often, a Spark job begins with HDFS. As a reminder, the Hadoop Distributed File System (HDFS) allows the storage and retrieval of data across multiple DataNodes. Creating an RDD leans upon HDFS to take advantage of parallel computing. Thus, a typical RDD begins by calling a large dataset from HDFS.

The Scala programming language and shell can be used to process large amounts of data in Apache Spark. Scala can be object-oriented and functional. Typically, it is also referred to as a statically typed language. While different preferences exist, RDDs can be developed easily with Scala, Java, or Python.

Spark documentation is well done and the basis for this lab. The credited/referenced tutorial lab is:

https://spark.apache.org/docs/latest/rdd-programming-guide.html#overview

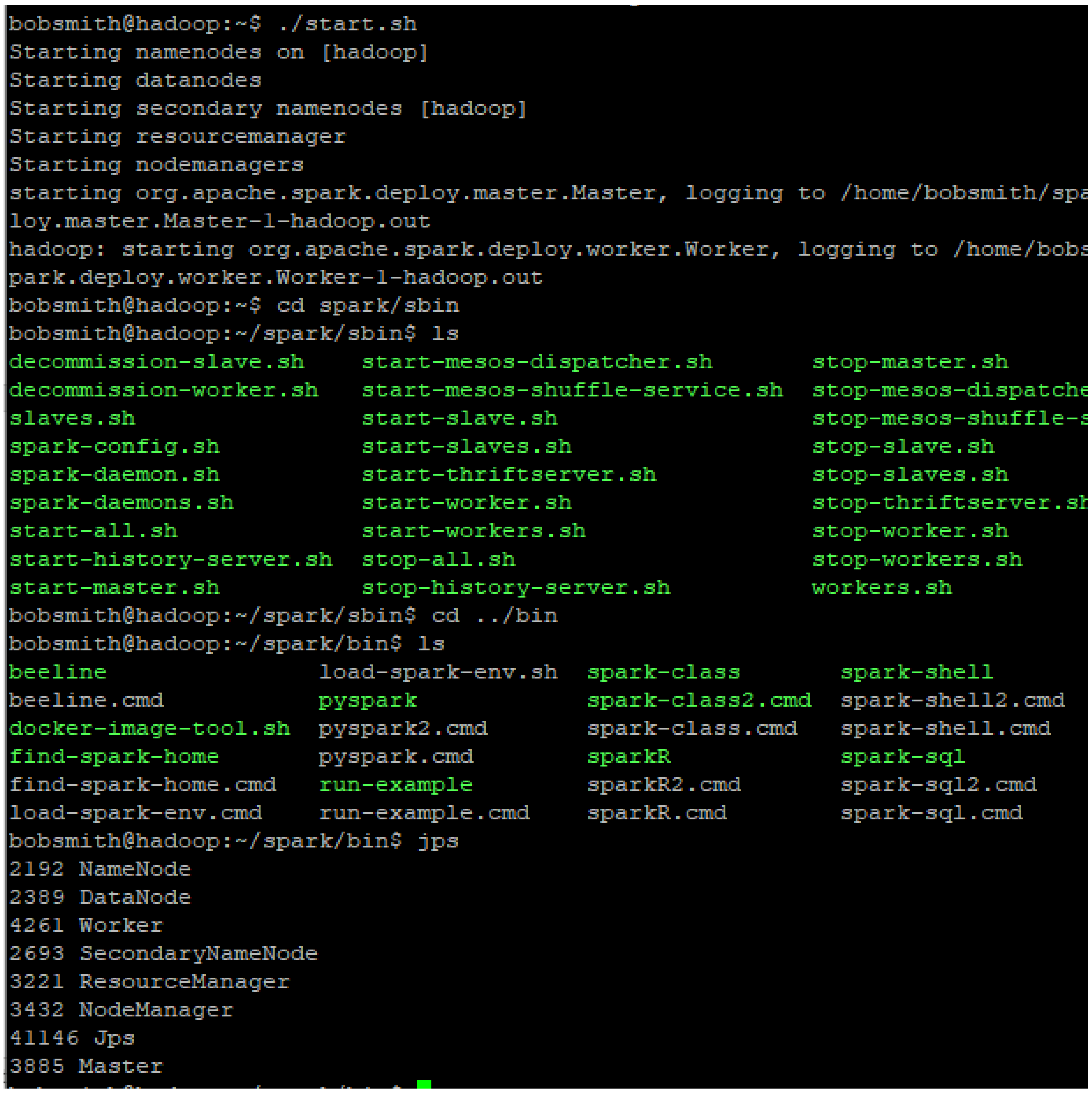
We also use our textbook for aspects of this lab.

Class Textbook: https://github.com/PacktPublishing/Big-Data-Architects-Handbook

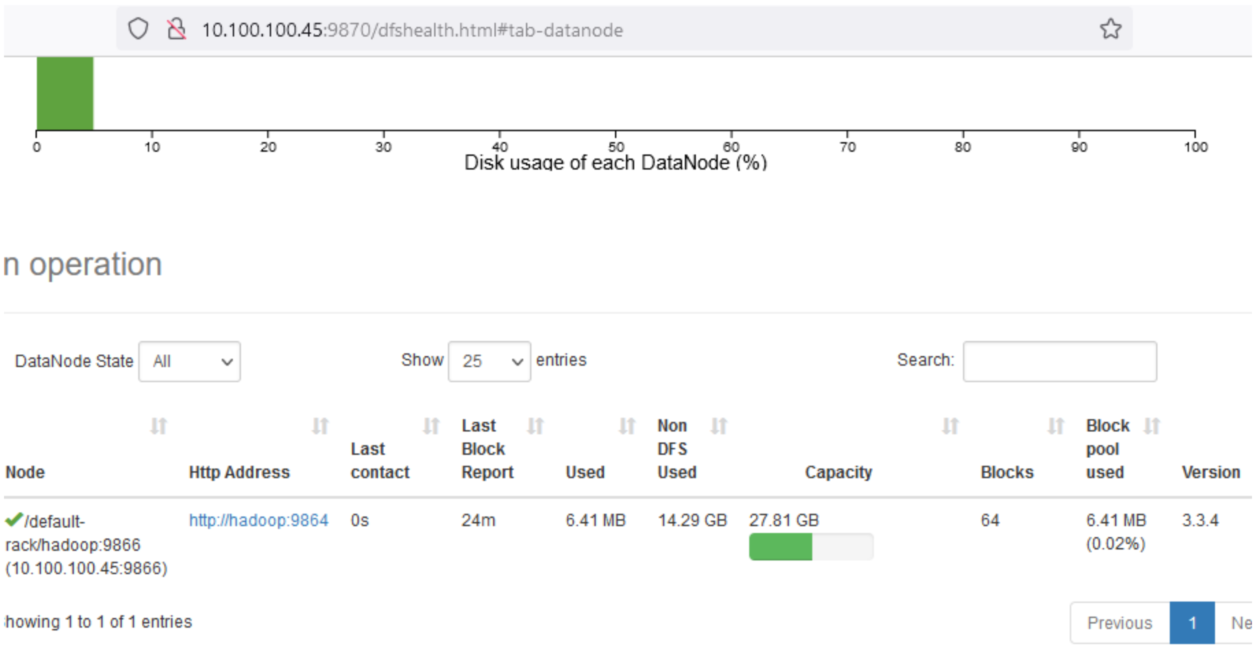
Lab Pre-Requisites

Before continuing this lab, it is assumed that you have a working Hadoop and Spark environment. Ensure you start Hadoop, HDFS, Yarn, and have at least one Spark worker running. As a reminder, the startup scripts are in the hadoop/sbin and spark/sbin directories. Go to those directories and start the services.

Once they are running, check them:



```
bobsmith@hadoop:~$ ./start.sh
Starting namenodes on [hadoop]
Starting datanodes
Starting secondary namenodes [hadoop]
Starting resourcemanager
Starting nodemanagers
starting org.apache.spark.deploy.master.Master, logging to /home/bobsmith/spa
loy.master.Master-1-hadoop.out
hadoop: starting org.apache.spark.deploy.worker.Worker, logging to /home/bobs
park.deploy.worker.Worker-1-hadoop.out
bobsmith@hadoop:~$ cd spark/sbin
bobsmith@hadoop:~/spark/sbin$ ls
decommission-slave.sh    start-mesos-dispatcher.sh       stop-master.sh
decommission-worker.sh   start-mesos-shuffle-service.sh  stop-mesos-dispatche
slaves.sh                start-slave.sh                  stop-mesos-shuffle-s
spark-config.sh          start-slaves.sh                 stop-slave.sh
spark-daemon.sh          start-thriftserver.sh           stop-slaves.sh
spark-daemons.sh         start-worker.sh                 stop-thriftserver.sh
start-all.sh             start-workers.sh                stop-worker.sh
start-history-server.sh  stop-all.sh                     stop-workers.sh
start-master.sh          stop-history-server.sh          workers.sh
bobsmith@hadoop:~/spark/sbin$ cd ../bin
bobsmith@hadoop:~/spark/bin$ ls
beeline                 load-spark-env.sh  spark-class        spark-shell
beeline.cmd             pyspark            spark-class2.cmd   spark-shell2.cmd
docker-image-tool.sh    pyspark2.cmd       spark-class.cmd    spark-shell.cmd
find-spark-home         pyspark.cmd        sparkR             spark-sql
find-spark-home.cmd     run-example        sparkR2.cmd        spark-sql2.cmd
load-spark-env.cmd      run-example.cmd    sparkR.cmd         spark-sql.cmd
bobsmith@hadoop:~/spark/bin$ jps
2192 NameNode
2389 DataNode
4261 Worker
2693 SecondaryNameNode
3221 ResourceManager
3432 NodeManager
41146 Jps
3885 Master
```


10.100.100.45:9870/dfshealth.html#tab-datanode

n operation

DataNode State All

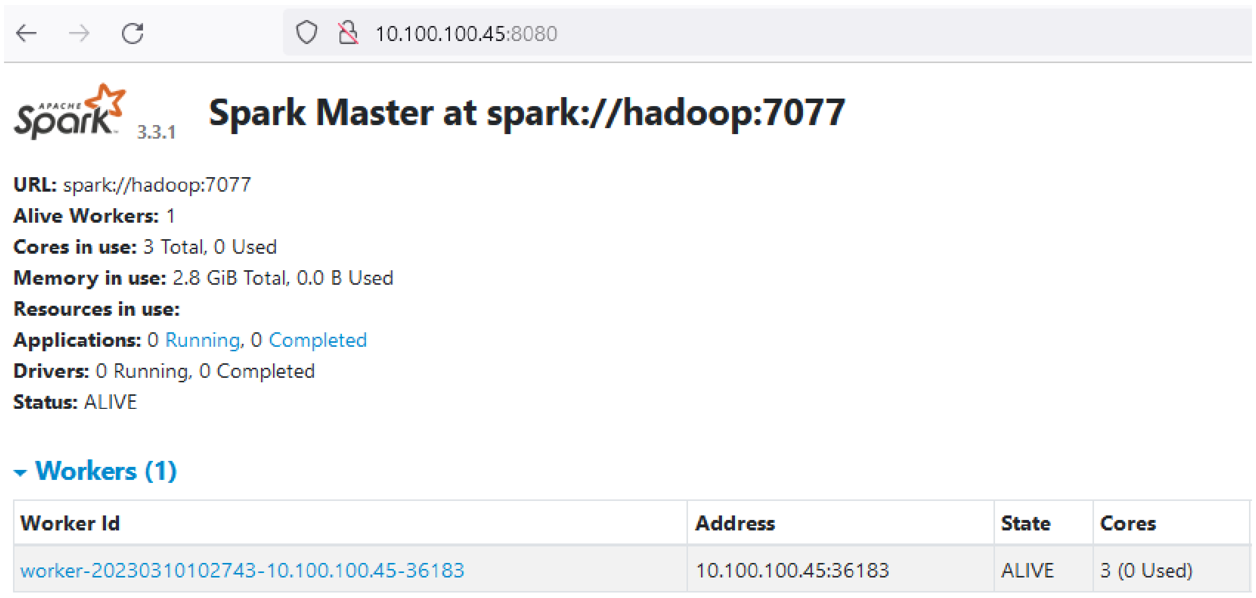
Show 25 entries

Search:

Node	Http Address	Last contact	Last Block Report	Used	Non DFS Used	Capacity	Blocks	Block pool used	Version
✓/default-rack/hadoop:9866 (10.100.100.45:9866)	http://hadoop:9864	0s	24m	6.41 MB	14.29 GB	27.81 GB	64	6.41 MB (0.02%)	3.3.4

howing 1 to 1 of 1 entries

Previous 1 Ne


10.100.100.45:8080

3.3.1 **Spark Master at spark://hadoop:7077**

URL: spark://hadoop:7077
Alive Workers: 1
Cores in use: 3 Total, 0 Used
Memory in use: 2.8 GiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (1)

Worker Id	Address	State	Cores
worker-20230310102743-10.100.100.45-36183	10.100.100.45:36183	ALIVE	3 (0 Used)

Introduction to Programming RDDs

Spark and RDDs are flexible and can use a variety of programming languages. Examples currently in the Spark documentation include Java, Python, and Scala. Spark applications in any programming language such as Java, Python, or Scala requires a main driver program to execute parallel functions on an HDFS cluster. RDDs call a dataset from HDFS and transform it using parallel computing. To do this, Spark relies upon shared variables.

A theme you will often see in all Spark capable programming languages is the need to broadcast data across all the HDFS DataNodes and then combine or collect the data from all of the HDFS DataNodes.

There are different types of shared variables in Spark but two you should initially understand are broadcast variables and accumulators. Broadcast variables cache values or variable contents in Spark across all HDFS nodes. Think of this like a TV or wireless signal that an antenna broadcasts out to many devices. Accumulators are able to perform sums and counters on the contents of variables that are broadcasted on all DataNodes. Often, it is necessary to combine or collect data from parallel DataNodes to perform operations like summations.

The primary class we use to create RDDs with accumulators and broadcast variables on a Spark cluster is SparkContext. See the constructors and values that can be used in this class here:

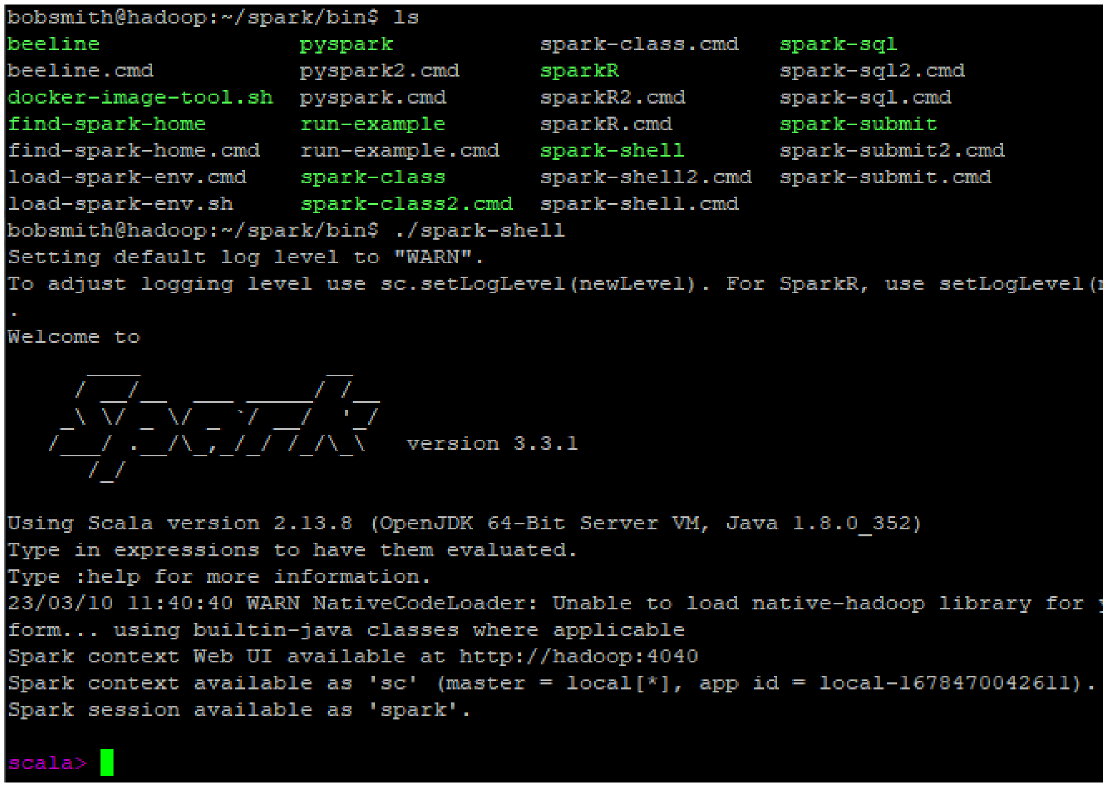
https://spark.apache.org/docs/latest/api/scala/org/apache/spark/SparkContext.html

A second class that is very helpful is SparkConf, which has constructors for key-value pairs. We need to establish the Spark Configuration with SparkConf including the primary HDFS server:

https://spark.apache.org/docs/latest/api/scala/org/apache/spark/SparkConf.html

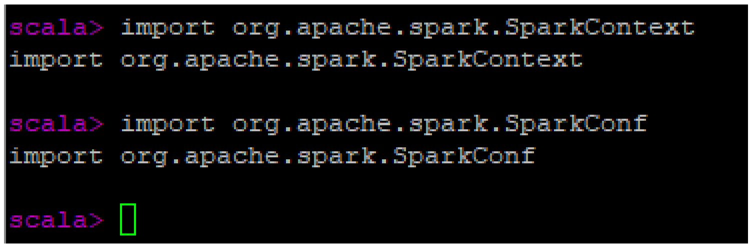
To get started in Sala, let's start by importing our classes. In Scala, this is very similar to Java. Open a Scala Shell and import the SparkContext and SparkConf classes.

Here is a screenshot:



```
bobsmith@hadoop:~/spark/bin$ ls
beeline                  pyspark            spark-class.cmd    spark-sql
beeline.cmd              pyspark2.cmd       sparkR             spark-sql2.cmd
docker-image-tool.sh     pyspark.cmd        sparkR2.cmd        spark-sql.cmd
find-spark-home          run-example        sparkR.cmd         spark-submit
find-spark-home.cmd      run-example.cmd    spark-shell        spark-submit2.cmd
load-spark-env.cmd       spark-class        spark-shell2.cmd   spark-submit.cmd
load-spark-env.sh        spark-class2.cmd   spark-shell.cmd
bobsmith@hadoop:~/spark/bin$ ./spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(
.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.3.1
      /_/

Using Scala version 2.13.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_352)
Type in expressions to have them evaluated.
Type :help for more information.
23/03/10 11:40:40 WARN NativeCodeLoader: Unable to load native-hadoop library for
form... using builtin-java classes where applicable
Spark context Web UI available at http://hadoop:4040
Spark context available as 'sc' (master = local[*], app id = local-1678470042611).
Spark session available as 'spark'.

scala>
```

Like Java, use the import key word to import classes you want to use, like this:



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

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

scala>
```

To learn some basic examples we will use the Spark Shell. Regardless, we typically use spark-submit to run large jobs across the entire Spark cluster. Here are a few examples from the Spark documentation:

# Run application locally on 8 cores

./bin/spark-submit \

  --class org.apache.spark.examples.SparkPi \

  --master local[8] \

  /path/to/examples.jar \

  100

```
# Run on a Spark standalone cluster in client deploy mode
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master spark://10.100.100.10:7077 \
  --executor-memory 20G \
  --total-executor-cores 100 \
  /path/to/examples.jar \
  1000
```

SparkContext needs an object to show Spark how to access the cluster. We can use a “local” string to run in local mode instead of a cluster or we can use YARN.

Here are a few examples:

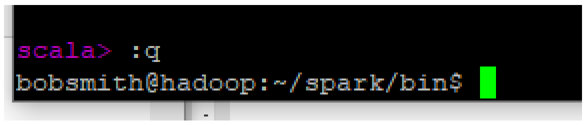
```
val conf = new SparkConf().setAppName(appName).setMaster(local);

new SparkContext(conf);
```

YARN:

```
val conf = new SparkConf().setAppName(appName).setMaster(yarn);

new SparkContext(conf);
```

Regardless, if you are using the Spark Shell, it typically creates a SparkContext by default using the variable “sc”.

Next, enter “:q” at the Scala prompt to quit Scala:



```
scala> :q
bobsmith@hadoop:~/spark/bin$
```

Creating an RDD

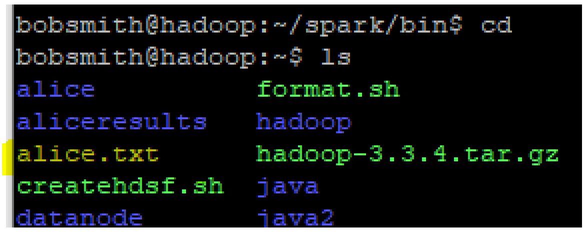
To put our Spark skills to the test, let's download some eBooks from Project Gutenberg:

https://www.gutenberg.org/

Feel free to download any eBook you would like to use for some example word count and key-value pair examples.

The book used in this tutorial is Alice's Adventures in Wonderland by Lewis Carroll. The plain text version is here: https://www.gutenberg.org/cache/epub/11/pg11.txt

nano alice.txt (or whatever book you would like to use) in your /home/username directory and save it with the text of the eBook. Here is a screenshot:

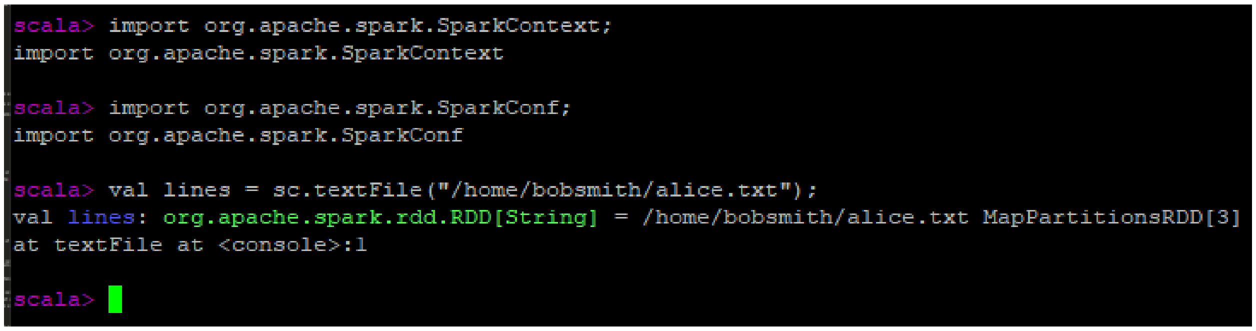


```
bobsmith@hadoop:~/spark/bin$ cd
bobsmith@hadoop:~$ ls
alice            format.sh
aliceresults     hadoop
alice.txt        hadoop-3.3.4.tar.gz
createhdsf.sh    java
datanode         java2
```

Next, we are going to create a new SparkContext using the Scala Shell with the Alice in Wonderland book or whatever book you would like to use. Alice and Wonderful happens to have many examples and tutorials so we will go with this for now. To declare a variable in Scala we can use the "val" keyword. Create a variable and pass it the contents of your eBook using the "sc" or SparkContext class:

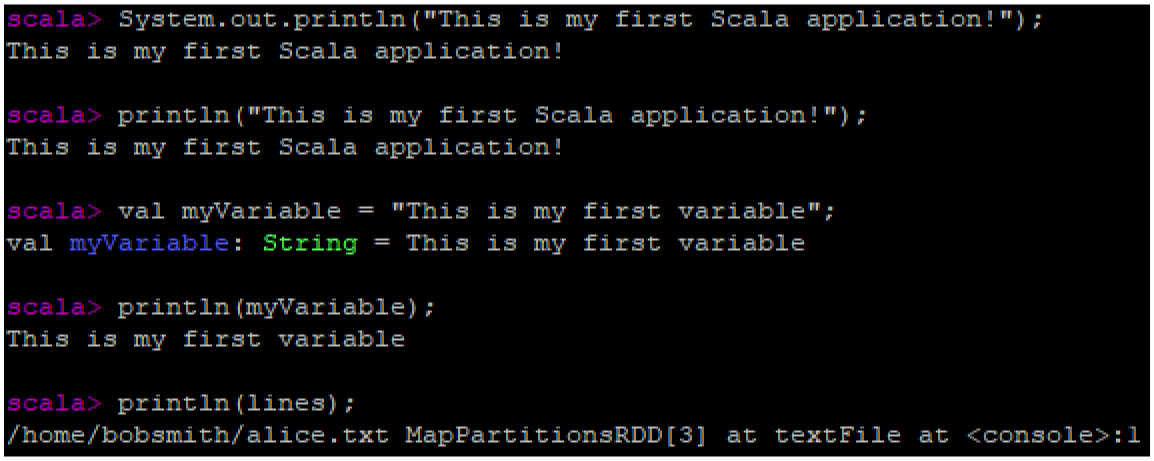
val lines = sc.textFile("/home/bobsmith/alice.txt");

Here is a screenshot:



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

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

scala> val lines = sc.textFile("/home/bobsmith/alice.txt");
val lines: org.apache.spark.rdd.RDD[String] = /home/bobsmith/alice.txt MapPartitionsRDD[3]
at textFile at <console>:1

scala>
```

In this example, I created the "lines" variable, which is a SparkContext with the contents of the Alice.txt file. I can now use the functions within the SparkContext class to do any number of functions on this data. I can also use other common classes and methods like in Java from the Scala shell. For example, you can use the System class or println to print the contents of a variable for troubleshooting. Here is a screenshot:

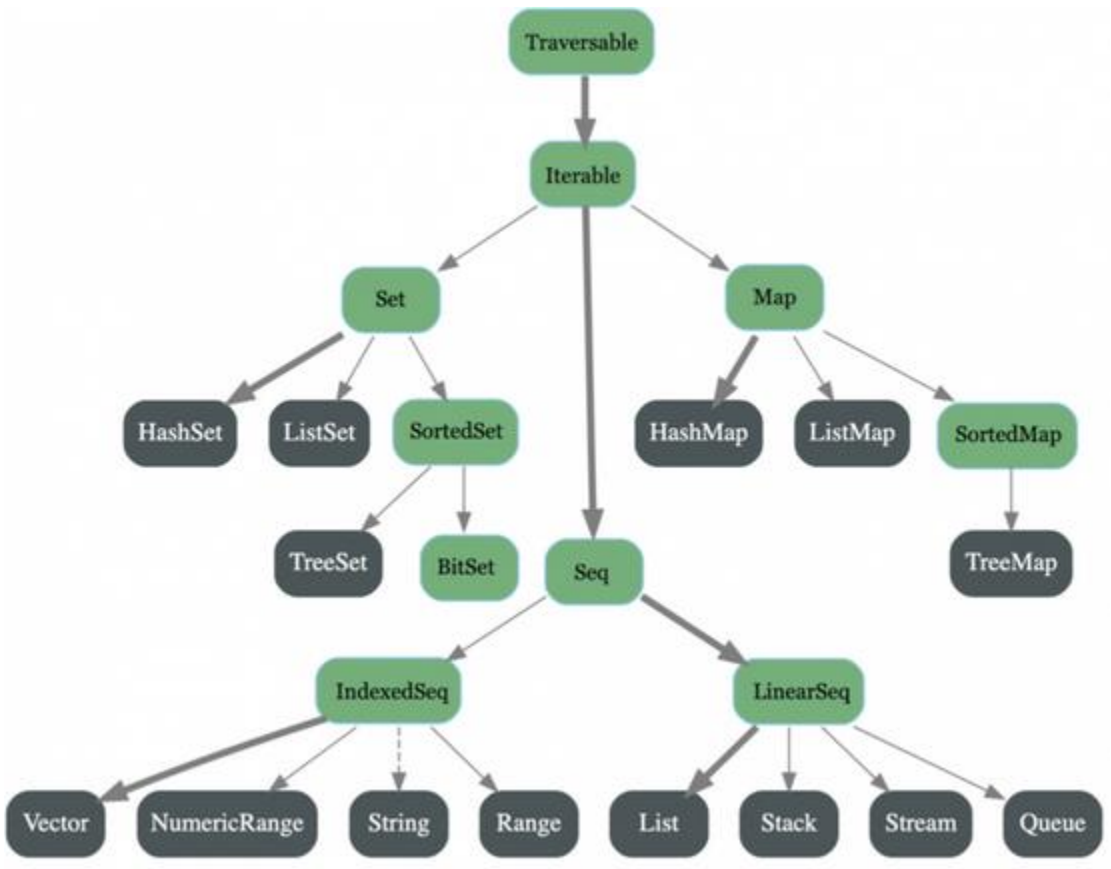


```
scala> System.out.println("This is my first Scala application!");
This is my first Scala application!

scala> println("This is my first Scala application!");
This is my first Scala application!

scala> val myVariable = "This is my first variable";
val myVariable: String = This is my first variable

scala> println(myVariable);
This is my first variable

scala> println(lines);
/home/bobsmith/alice.txt MapPartitionsRDD[3] at textFile at <console>:1
```

At this point, ensure your new sc variable has your eBook contents in it using a println.

Collections in Scala

To emulate a basic key value pair, we can creating a map in Scala. A collection in Scala is a data structure which holds a group of objects. Examples of collections include Maps, Arrays, Lists, Sets, and many others. Here is an example of the Collection Hierarchy from https://www.baeldung.com/scala/collections:



The Figure is from https://www.baeldung.com/scala/collections

We can apply transformations to these collections using several methods. A very popular method offered by Scala is map(). Every collection object has the map() method. Like most other Collections, creating a map allows us to perform other tasks very easily like sorting, searching, etc.

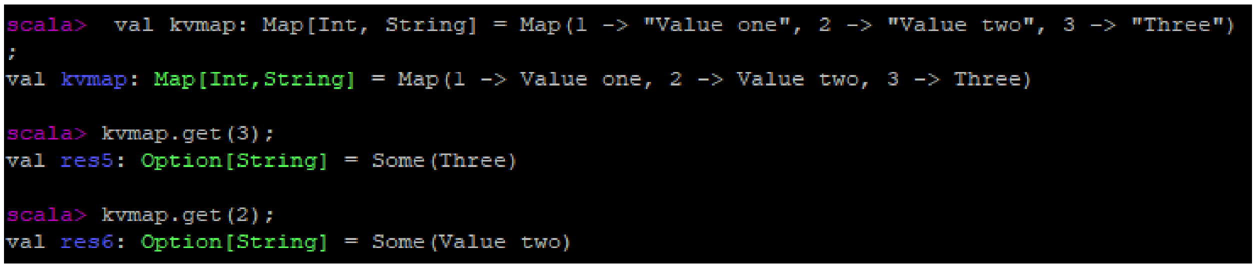
Let’s try an example in Scala. Create a “kvmap” variable and give it a few key-value pairs:

```
val kvmap: Map[Int, String] = Map(1 -> "Value one", 2 -> "Value two", 3 -> "Three");
```

Next, use your new map variable to try some different functions on the KV pairs. For example, try to return the third key or “Three”. The code simply can use the map methods like get:

```
kvmap.get(3);
```

Here is a screenshot:



```
scala>  val kvmap: Map[Int, String] = Map(1 -> "Value one", 2 -> "Value two", 3 -> "Three")
;
val kvmap: Map[Int,String] = Map(1 -> Value one, 2 -> Value two, 3 -> Three)

scala> kvmap.get(3);
val res5: Option[String] = Some(Three)

scala> kvmap.get(2);
val res6: Option[String] = Some(Value two)
```

Collections in Pyspark

As we have previously noted, we can also use Python just as easily. Thus, for those more familiar with Python, try to create a similar application using pyspark. Quit Scala and enter pyspark. You will need to use an anonymous function or lambda in Python to process certain functions. Thus, if necessary refresh your Python skills by practicing these functions.

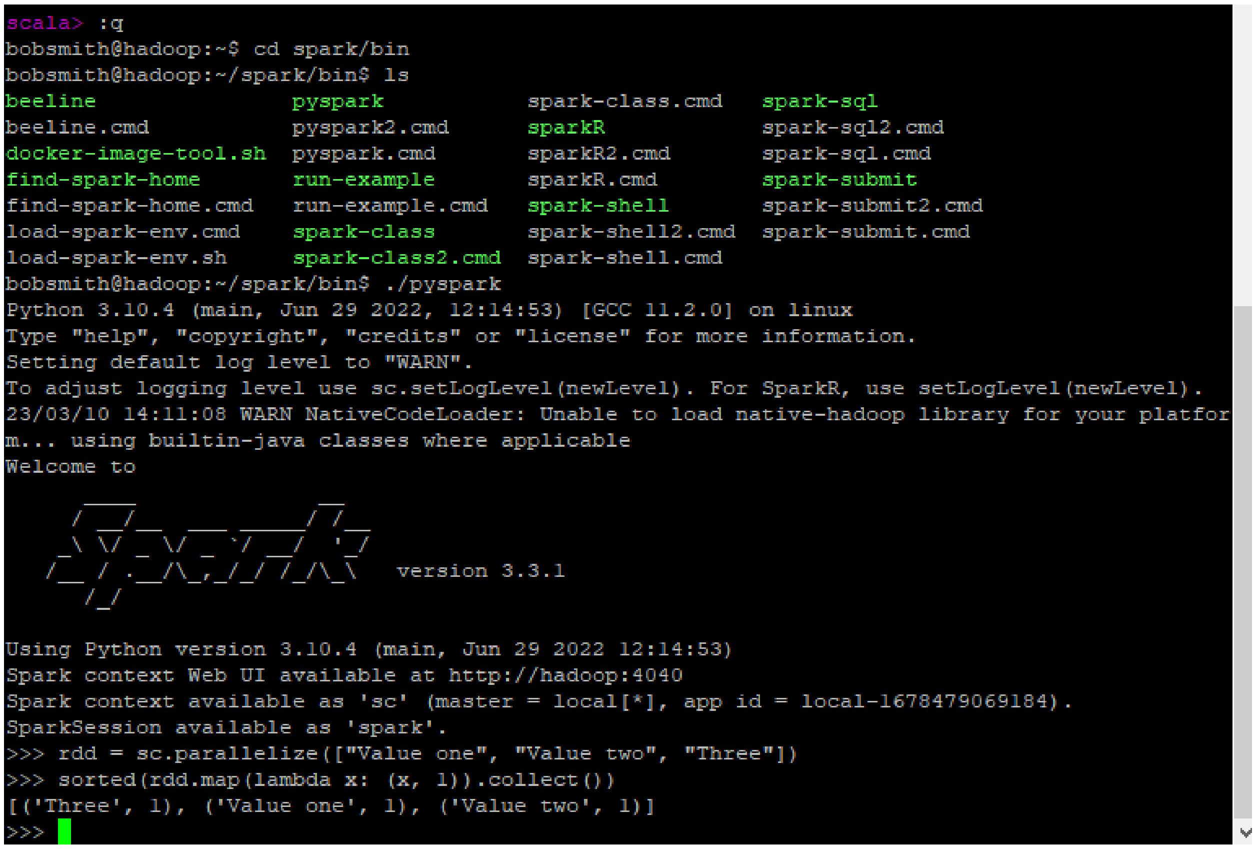
For other practice, we will try a different SparkContext function such as parallelized collections. The parallelize method uses the driver program to distribute the dataset in parallel DataNodes. Thus, we can use parallelize to create the KV in an HDFS environment. Then, we can use the map class similar to Scala. For example:

https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.RDD.map.html

rdd = sc.parallelize(["Value one", "Value two", "Three"])

sorted(rdd.map(lambda x: (x, 2)).collect())

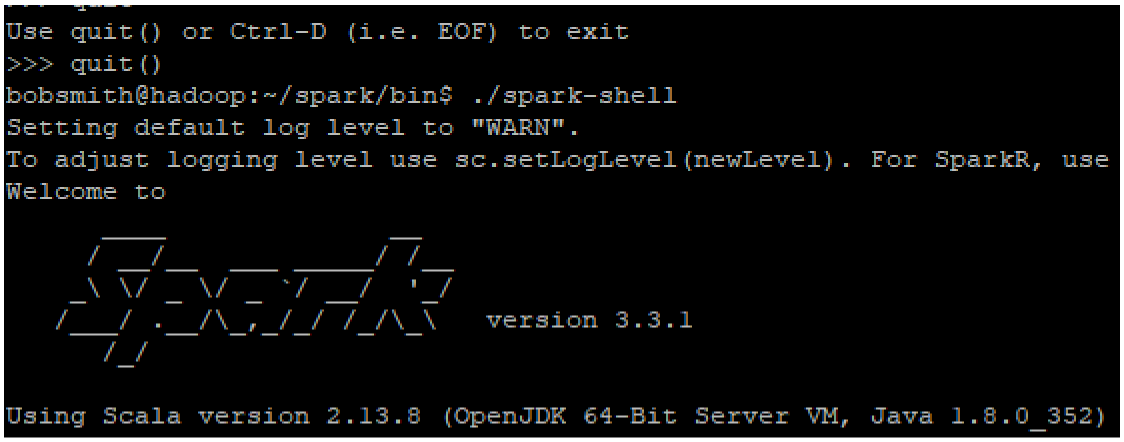
Here is a screenshot:



```
scala> :q
bobsmith@hadoop:~$ cd spark/bin
bobsmith@hadoop:~/spark/bin$ ls
beeline                  pyspark            spark-class.cmd    spark-sql
beeline.cmd              pyspark2.cmd       sparkR             spark-sql2.cmd
docker-image-tool.sh     pyspark.cmd        sparkR2.cmd        spark-sql.cmd
find-spark-home          run-example        sparkR.cmd         spark-submit
find-spark-home.cmd      run-example.cmd    spark-shell        spark-submit2.cmd
load-spark-env.cmd       spark-class        spark-shell2.cmd   spark-submit.cmd
load-spark-env.sh        spark-class2.cmd   spark-shell.cmd
bobsmith@hadoop:~/spark/bin$ ./pyspark
Python 3.10.4 (main, Jun 29 2022, 12:14:53) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/03/10 14:11:08 WARN NativeCodeLoader: Unable to load native-hadoop library for your platfor
m... using builtin-java classes where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.3.1
      /_/

Using Python version 3.10.4 (main, Jun 29 2022 12:14:53)
Spark context Web UI available at http://hadoop:4040
Spark context available as 'sc' (master = local[*], app id = local-1678479069184).
SparkSession available as 'spark'.
>>> rdd = sc.parallelize(["Value one", "Value two", "Three"])
>>> sorted(rdd.map(lambda x: (x, 1)).collect())
[('Three', 1), ('Value one', 1), ('Value two', 1)]
>>>
```

RDD Key-Value Pairs

Getting back to Scala to finish our KV example, quit pyspark by entering “>>> quit()” and open the scala-shell again.



```
Use quit() or Ctrl-D (i.e. EOF) to exit
>>> quit()
bobsmith@hadoop:~/spark/bin$ ./spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.3.1
      /_/

Using Scala version 2.13.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_352)
```

Go ahead and import the SparkContext and SparkConf classes:

import org.apache.spark.SparkContext;

import org.apache.spark.SparkConf;

Next, create a SparkContext again using your eBook file:

val lines = sc.textFile("/home/bobsmith/alice.txt");

Subsequently, now that we have a basic understanding of the Map method, let’s use it to create KV pairs from our eBook and perform a transformation. As an example, pick a common word in your eBook and use this to create a KV pair. I will pick the word “Alice” since it is common in Alice and Wonderland. And, I will simply add an integer of “1” to every instance of Alice in the pair:

val pairs = lines.map(alice => (alice, 1));

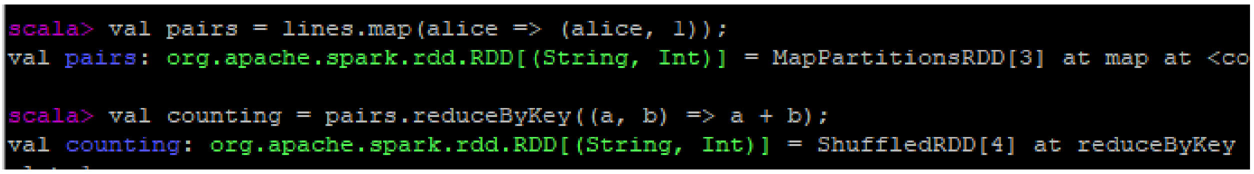
To see the pairs we can loop through the map with the foreach method:

pairs.foreach(println);

RDD Transformations

Another basic example is to count all of the KV pairs. To add all the values with the same key you can use the “reduceByKey” method like this:

val counting = pairs.reduceByKey((a, b) => a + b);



```
scala> val pairs = lines.map(alice => (alice, 1));
val pairs: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[3] at map at <co

scala> val counting = pairs.reduceByKey((a, b) => a + b);
val counting: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[4] at reduceByKey
```

A method we have not used yet that is perfect for reduceByKey is “collect”. Collect allows use to combine all summed values but is also valuable for other parallel operations that need to combine any results, including filters.

counting.collect;

Similarly, we can use collect with other unique methods like sortByKey, which sorts your pairs by the key alphabetically. For example:

counting.sortByKey().collect;

Another helpful function is to work with filters to filter various elements in the dataset. For example, to filter all sentences that start with Alice:

lines.filter(_.startsWith("Alice")).collect;

To filter for any cats that exist in the eBook:

lines.filter(_.contains("cat")).collect;

Note, an underscore in Scala is a wildcard for matching unknown patterns. We can also use it to ignore a value in a method or function. For example:

val myMapVariable = (1 to 3).map(_ => "one");

Using the underscore, we ignored whatever value we have in the *map* anonymous function. Essentially, we can also use the anonymized parameter as a placeholder in function when we do not want to set a static parameter. It is more generic and gives us more freedom when using Scala functions.

RDDs in a Cluster

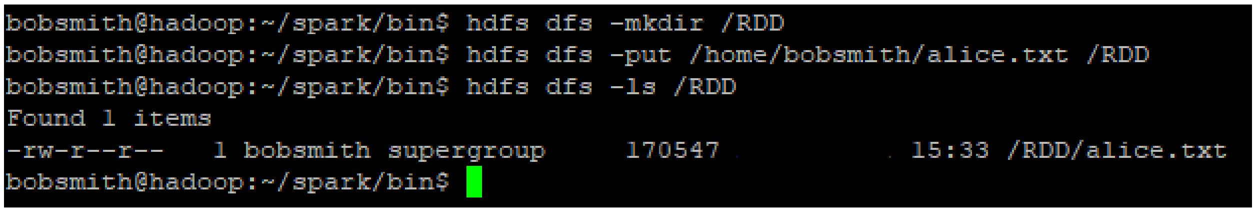
Many options exist when running RDDs in a cluster. To see current documentation:

https://spark.apache.org/docs/latest/submitting-applications.html

Reference the documentation here if you are running YARN:

https://spark.apache.org/docs/latest/running-on-yarn.html

To run RDDs in a cluster, we need to leverage our HDFS environment. If you have not already, quit Scala and place an eBook in a new RDD HDFS directory. Here is an example screenshot:

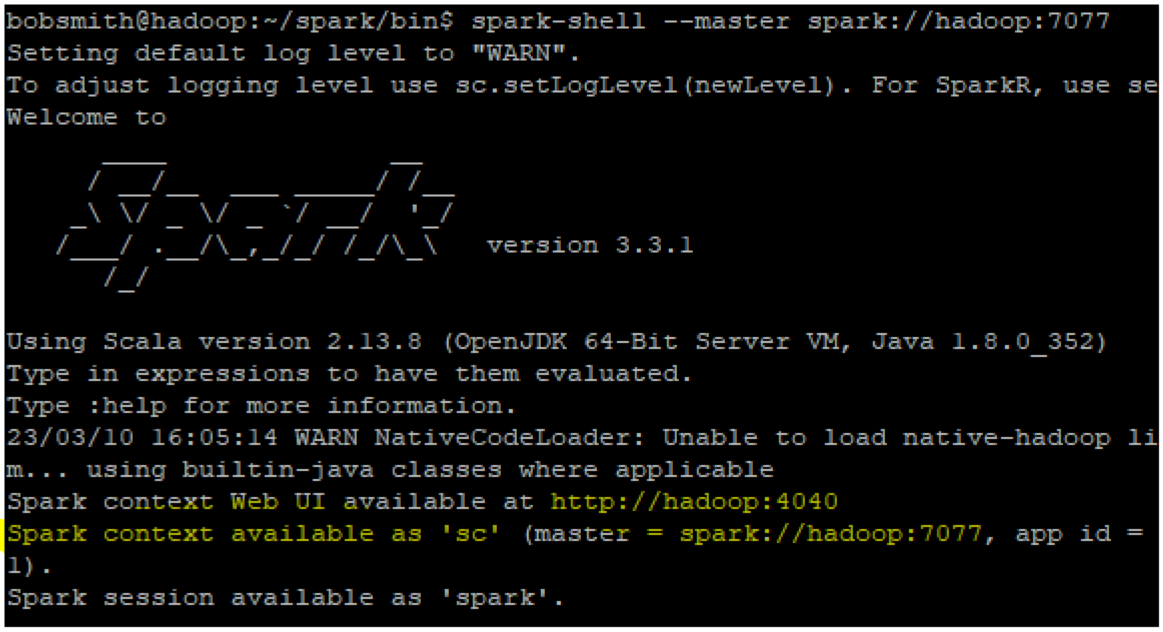


```
bobsmith@hadoop:~/spark/bin$ hdfs dfs -mkdir /RDD
bobsmith@hadoop:~/spark/bin$ hdfs dfs -put /home/bobsmith/alice.txt /RDD
bobsmith@hadoop:~/spark/bin$ hdfs dfs -ls /RDD
Found 1 items
-rw-r--r--   1 bobsmith supergroup     170547            15:33 /RDD/alice.txt
bobsmith@hadoop:~/spark/bin$
```

We can run Scala jobs using compiled jar files but we can also open Scala using the Spark cluster with our hostname URL like this:

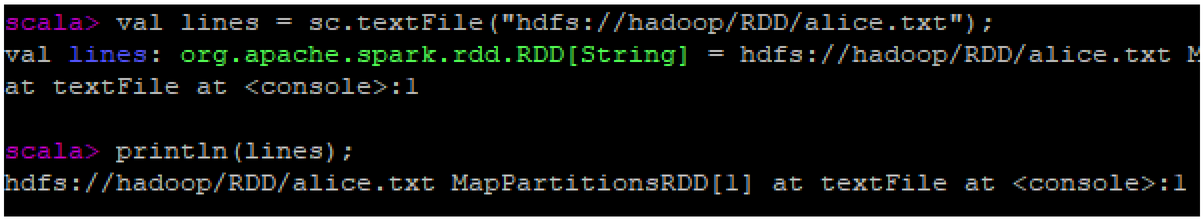
spark-shell --master spark://hadoop:7077

Here is a screenshot as an example:



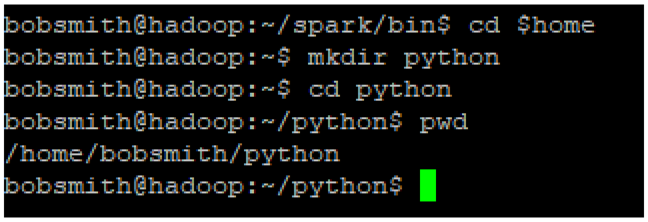
```
bobsmith@hadoop:~/spark/bin$ spark-shell --master spark://hadoop:7077
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use se
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.3.1
      /_/

Using Scala version 2.13.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_352)
Type in expressions to have them evaluated.
Type :help for more information.
23/03/10 16:05:14 WARN NativeCodeLoader: Unable to load native-hadoop li
m... using builtin-java classes where applicable
Spark context Web UI available at http://hadoop:4040
Spark context available as 'sc' (master = spark://hadoop:7077, app id =
1).
Spark session available as 'spark'.
```

We can now reference our eBook from our HDFS URL. Here is a screenshot example:



```
scala> val lines = sc.textFile("hdfs://hadoop/RDD/alice.txt");
val lines: org.apache.spark.rdd.RDD[String] = hdfs://hadoop/RDD/alice.txt M
at textFile at <console>:1

scala> println(lines);
hdfs://hadoop/RDD/alice.txt MapPartitionsRDD[1] at textFile at <console>:1
```

Next, to run a Python example create a new python folder in your home directory:



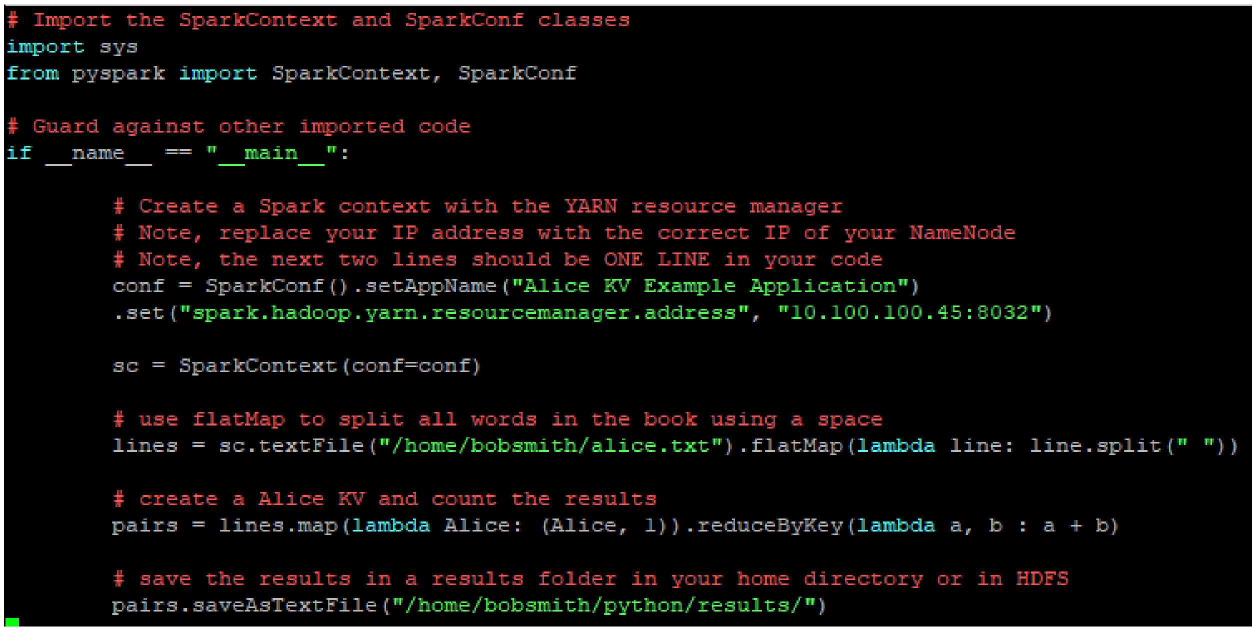
```
bobsmith@hadoop:~/spark/bin$ cd $home
bobsmith@hadoop:~$ mkdir python
bobsmith@hadoop:~$ cd python
bobsmith@hadoop:~/python$ pwd
/home/bobsmith/python
bobsmith@hadoop:~/python$
```

Before we can create a Python application, we need to understand how to use SparkConf to configure Spark to run our application. As a basic example, the YARN configuration looks approximately like this (you may need to change this depending on your own environment):

conf = SparkConf().setAppName("Application paName").set("spark.hadoop.yarn.resourcemanager.address", "10.100.100.45:8032")

Note, you need to change the IP address to your correct IP or your hostname.

To test a Python application let's import our eBook, perform a basic split using a space, and count the words. Create a basic Python application and name it example.py. Example code:



```
# Import the SparkContext and SparkConf classes
import sys
from pyspark import SparkContext, SparkConf

# Guard against other imported code
if __name__ == "__main__":

        # Create a Spark context with the YARN resource manager
        # Note, replace your IP address with the correct IP of your NameNode
        # Note, the next two lines should be ONE LINE in your code
        conf = SparkConf().setAppName("Alice KV Example Application")
        .set("spark.hadoop.yarn.resourcemanager.address", "10.100.100.45:8032")

        sc = SparkContext(conf=conf)

        # use flatMap to split all words in the book using a space
        lines = sc.textFile("/home/bobsmith/alice.txt").flatMap(lambda line: line.split(" "))

        # create a Alice KV and count the results
        pairs = lines.map(lambda Alice: (Alice, 1)).reduceByKey(lambda a, b : a + b)

        # save the results in a results folder in your home directory or in HDFS
        pairs.saveAsTextFile("/home/bobsmith/python/results/")
```

Alternatively, to use Spark as the resource manager rather than YARN, you have to use the .setMaster method. Here is an example where "hostname" is your server name:

conf = SparkConf().setAppName("Application Name").setMaster("spark://hostname:7077")

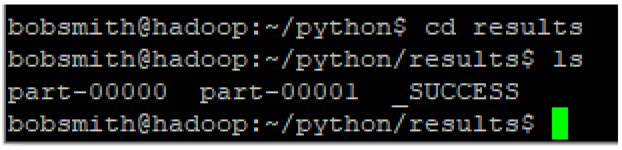
Also, instead of accessing local files, you can attempt to access HDFS. Review your core-site.xml file in hadoop/etc/hadoop/core-site.xml. Look for the <property> for fs.default.name, which should give you your correct hdfs:// value. Here is an example:

sc.textFile("hdfs://hostname:9000/alice.txt")

Next, run the code using spark-submit:

```
spark-submit example.py
```

Check your logs and ensure it ran successfully. It should have created a results folder per the last line of code. Here is an example screenshot:



```
bobsmith@hadoop:~/python$ cd results
bobsmith@hadoop:~/python/results$ ls
part-00000  part-00001  _SUCCESS
bobsmith@hadoop:~/python/results$
```

This concludes our introduction to RDDs. I hope you are enjoying the labs and wanting to learn more!