Guidelines
For
Hadoop and Spark Cluster Usage

Procedure to create an account in CSX.

If you are taking a CS prefix course, you already have an account; to get an initial password created:

1. Login to https://cs.okstate.edu/pwreset to set the password for CSX.

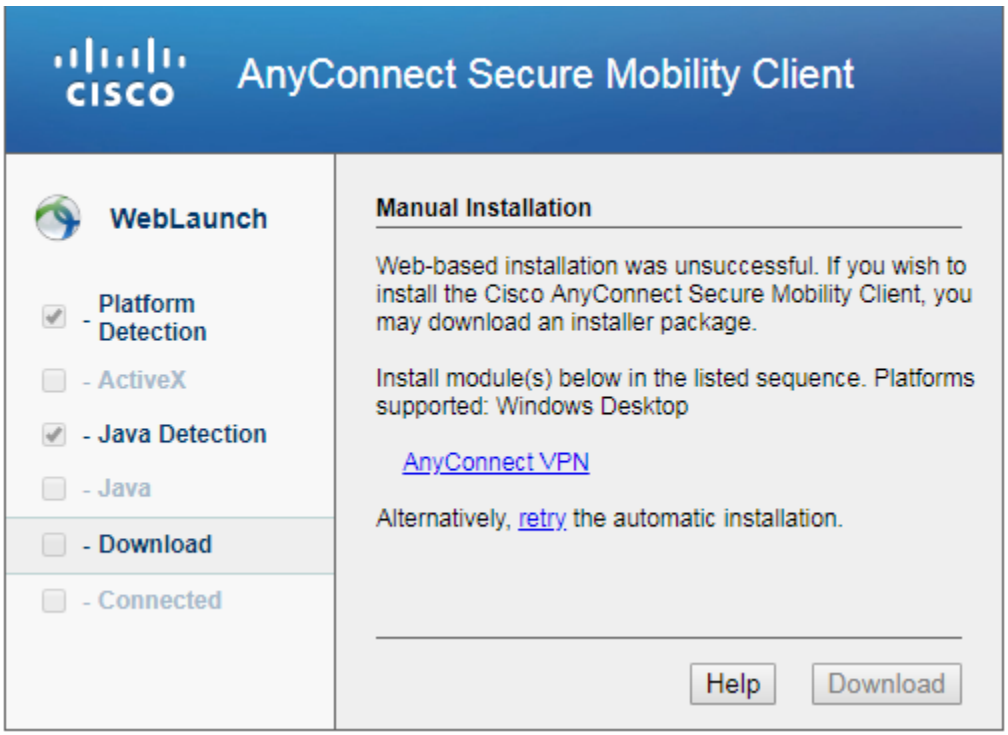
Putty Download:

Windows users use the below link to download PUTTY.

https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

Connecting from outside the university:

1. Login to **https://osuvpn.okstate.edu/+CSCOE+/logon.html** with OKEY username and password
2. You will be redirected to the below screenshot go to download section and download AnyConnectVPN
3. Connect every time when you want to use the servers in CS Department.



4. After Connecting open the putty and in Hostname give csh.cs.okstae.edu for CSH login cluster, set port as 22 and connection Type as SSH.

Reference:

https://www.cs.okstate.edu/loggingon.html

Login Procedure to Hadoop1:

1. Login with your user name (CSX user name) to csh.cs.okstate.edu

2. MAC/LINUX users type ssh user@csh.cs.okstate.edu from your terminal. Windows users can use Putty to login to CSH.

3. Enter your CSX password when asked.

4. Then login into bbod by typing ssh user@bbod

5. Enter your CSX password when asked.

6. After logging onto bbod, logon to master node of our Hadoop cluster by typing ssh user@hadoop1

7. Enter your CSX password when asked.

8. You will have to access only the folder /user_name in HDFS.

9. You can use Hadoop in the cluster.

Process to collect Twitter Data:

Apache Flume is an open-source software that helps to store the streaming data into HDFS. A flume agent should be created through which we can stream the data. The following steps describe the process to collect the twitter data.

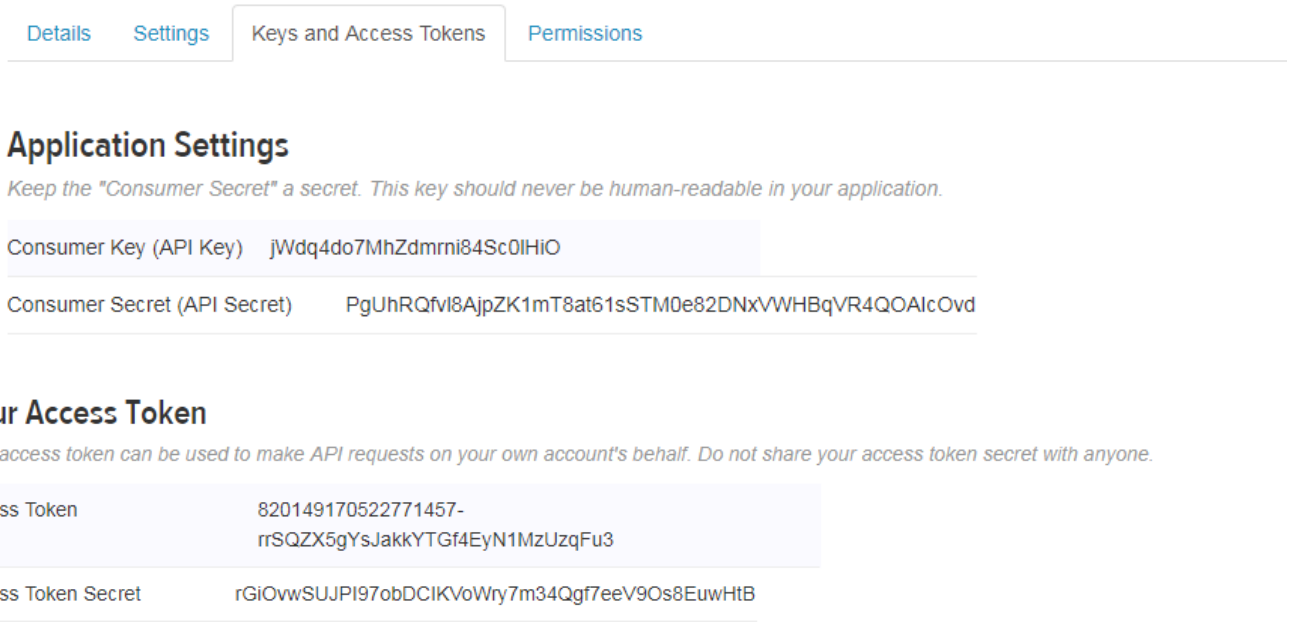
1. Create an account in twitter and login with the credentials.

2. Navigate to **https://apps.twitter.com/** and create a new app.

3. Give the Name and the Description on what are you the data, and the website URL is **https://twitter.com/**

4. Get the consumer secret, consumer token, access token and access token secret for your application. To get the consumer secret, consumer token:

> (a)Go to Keys and Access Tokens get the consumer key and consumer secret, and then go to Your Access tokens and get the access tokens and access token secret.



4. There are 3 components for a twitter agent namely source, sink and channel.

5. The flume source connects to Twitter API and receives data in JSON format which in turn stored into HDFS.

6. Now, create a configuration file for the flume agent by specifying the consumer key, consumer secret, access token, access token secret, keywords and HDFS path.

A sample configuration file with file extension .conf is shown below. It shows all the keys and keywords to be used to collect the twitter data.

```
TwitterAgent.sources = Twitter
TwitterAgent.channels = MemChannel
TwitterAgent.sinks = HDFS
TwitterAgent.sources.Twitter.type = com.cloudera.flume.source.TwitterSource
TwitterAgent.sources.Twitter.channels = MemChannel
TwitterAgent.sources.Twitter.consumerKey =
TwitterAgent.sources.Twitter.consumerSecret =
TwitterAgent.sources.Twitter.accessToken =
TwitterAgent.sources.Twitter.accessTokenSecret =
TwitterAgent.sources.Twitter.keywords = Keywords to be specified here
TwitterAgent.sinks.HDFS.channel = MemChannel
TwitterAgent.sinks.HDFS.type = hdfs
TwitterAgent.sinks.HDFS.hdfs.path = Configuration File Path to store the data. #
hdfs://hadoop1:9000/rramine/Food_data/
TwitterAgent.sinks.HDFS.hdfs.fileType = DataStream
TwitterAgent.sinks.HDFS.hdfs.writeFormat = Text
TwitterAgent.sinks.HDFS.hdfs.batchSize = 100
TwitterAgent.sinks.HDFS.hdfs.rollSize = 0
TwtterAgent.sinks.HDFS.hdfs.rollCount = 0
TwitterAgent.channels.MemChannel.type = memory
TwitterAgent.channels.MemChannel.capacity = 10000
TwitterAgent.channels.MemChannel.transactionCapacity = 10000
```

Command to start the flume agent:

nohup $FLUME_HOME/bin/flume-ng agent -n TwitterAgent -f Configuration File Path &

Example: nohup $FLUME_HOME/bin/flume-ng agent -n TwitterAgent –f /autohome/rramine/bitcoin.conf &

nohup will make sure the data collection process runs continuously at the backend. nohup.out file is the log file that will be created as we start the process. The data collected will be in JSON format.

Command to check the count of the files.

hdfs dfs –count /username/(folder name) there is a limit on the number of files. No more data is put into files when the limit is reached.

Reference:

https://acadgild.com/blog/streaming-twitter-data-using-flume/

https://www.tutorialspoint.com/apache_flume/fetching_twitter_data.htm

Map Reduce Example Program:

Map Reduce is a framework using which we can process applications with huge amount of data in parallel.

An Example word count program in MapReduce:

Program: wordcount.java

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
public class wordcount {

public static class MapForWordCount extends Mapper<LongWritable, Text, Text, IntWritable>{
public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
{
String line = value.toString();
String[] words=line.split(" ");
for(String word: words )
{
    Text outputKey = new Text(word.toUpperCase().trim());
 IntWritable outputValue = new IntWritable(1);
 context.write(outputKey, outputValue);
}
}
}
public static class ReduceForWordCount extends Reducer<Text, IntWritable, Text, IntWritable>
{
public void reduce(Text word, Iterable<IntWritable> values, Context context) throws IOException,
InterruptedException
{
int sum = 0;
  for(IntWritable value : values)
  {
  sum += value.get();
  }
  context.write(word, new IntWritable(sum));
}
}
```

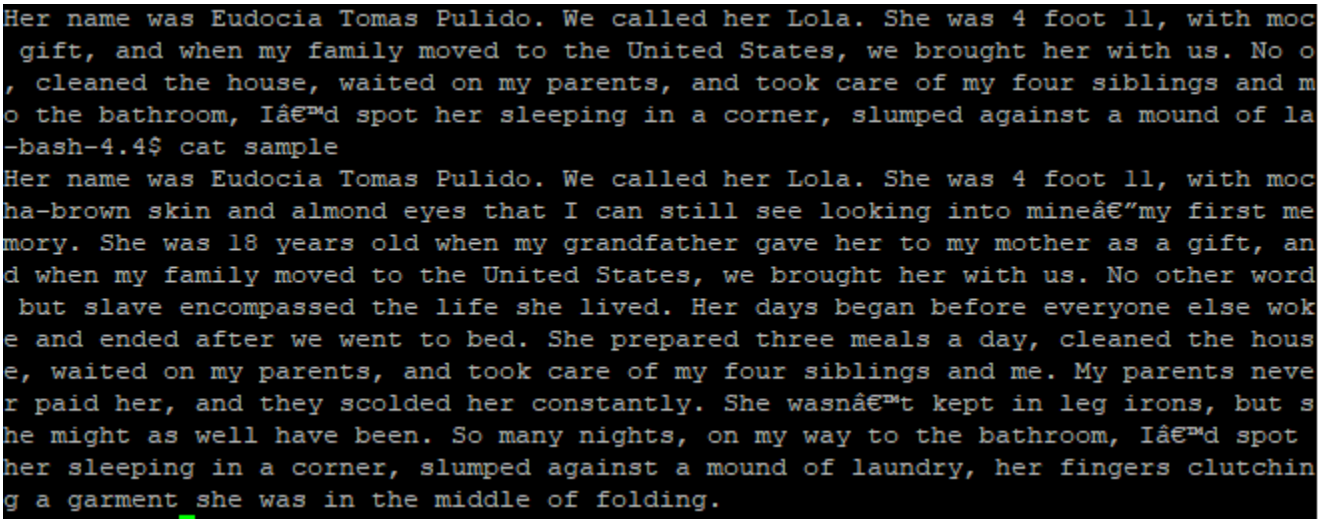
```
public static void main(String [] args) throws Exception
{
Configuration conf = new Configuration();
                Job job = Job.getInstance(conf, "Wordcount");
    job.setJarByClass(wordcount.class);
    job.setMapperClass(MapForWordCount.class);
    job.setReducerClass(ReduceForWordCount.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
                job.setMapOutputKeyClass(Text.class);
                job.setMapOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
```

Steps to execute MapReduce Programs:

1. Login to Hadoop1 and create a java file with program_name.java
2. Then compile the java program to get the class file
   Command: javac programname.java
3. Creation of jar file
   Command: jar –cvf program_name.jar program_name*.class
4. Command to run the Hadoop program
   Command: hadoop jar program_name.jar program_name hdfs://hadoop1:9000/inputpath hdfs://hadoop1:9000/outputpath
5. **Command to check output:** hdfs dfs -cat /outputpath/part-r-00000

The input file should be in hdfs.

Screenshot of input data:



```
Her name was Eudocia Tomas Pulido. We called her Lola. She was 4 foot 11, with moc
 gift, and when my family moved to the United States, we brought her with us. No o
, cleaned the house, waited on my parents, and took care of my four siblings and m
o the bathroom, Iâ€™d spot her sleeping in a corner, slumped against a mound of la
-bash-4.4$ cat sample
Her name was Eudocia Tomas Pulido. We called her Lola. She was 4 foot 11, with moc
ha-brown skin and almond eyes that I can still see looking into mineâ€”my first me
mory. She was 18 years old when my grandfather gave her to my mother as a gift, an
d when my family moved to the United States, we brought her with us. No other word
 but slave encompassed the life she lived. Her days began before everyone else wok
e and ended after we went to bed. She prepared three meals a day, cleaned the hous
e, waited on my parents, and took care of my four siblings and me. My parents neve
r paid her, and they scolded her constantly. She wasnâ€™t kept in leg irons, but s
he might as well have been. So many nights, on my way to the bathroom, Iâ€™d spot
her sleeping in a corner, slumped against a mound of laundry, her fingers clutchin
g a garment she was in the middle of folding.
```

Screenshot of output data:

```
11,	1
18	1
4	1
A	5
AFTER	1
AGAINST	1
ALMOND	1
AND	6
AS	2
BATHROOM,	1
BED.	1
BEEN.	1
BEFORE	1
BEGAN	1
BROUGHT	1
BUT	2
CALLED	1
CAN	1
CARE	1
CLEANED	1
CLUTCHING	1
CONSTANTLY.	1
CORNER,	1
DAY,	1
DAYS	1
ELSE	1
ENCOMPASSED	1
ENDED	1
EUDOCIA	1
EVERYONE	1
EYES	1
FAMILY	1
FINGERS	1
FIRST	1
FOLDING.	1
FOOT	1
FOUR	1
GARMENT	1
GAVE	1
GIFT,	1
GRANDFATHER	1
HAVE	1
HER	8
HER,	1
HOUSE,	1
```

Reference:

https://dzone.com/articles/word-count-hello-word-program-in-mapreduce

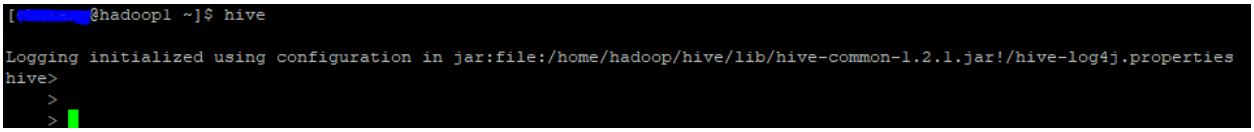
Apache Hive

Hive is a data warehousing package/infrastructure built on top of Hadoop. It provides an SQL dialect, called Hive Query Language(HQL) for querying data stored in a Hadoop cluster.

Feature	Description
Familiar	Query data with a SQL-based language
Fast	Interactive response times, even over huge datasets
Scalable and Extensible	As data variety and volume grows, more commodity machines can be added, without a corresponding reduction in performance
Compatible	Works with traditional data integration and data analytics tools.

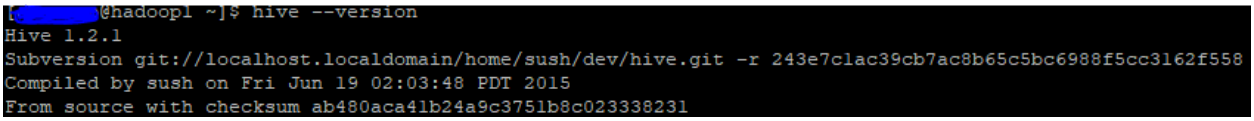
Please follow below steps to use Hive in the Hadoop Cluster:

First login to the Hadoop cluster (hadoop1) and type 'hive' on the console. A hive console will be initialized as shown in the below screenshot.



To verify the version of the hive, we can use below command

hive --version



Load Data to a Table in Hive:

We can load the data into Hive either from a local file or from HDFS. Below examples show how to load the data into hive from both local file and hdfs.

To load the data from a local file following query can be used:

LOAD DATA LOCAL INPATH ' MOCK_DATA.txt' INTO table local_table;

Below screenshots show how to create a table and load data from local file in hive.

```
[ @hadoop1 ~]$ hive

Logging initialized using configuration in jar:file:/home/
hive> CREATE TABLE local_table(id INT, name STRING)
    > Row format delimited
    > Fields terminated by '\t';
OK
Time taken: 1.276 seconds
```

```
hive>
    > LOAD DATA LOCAL INPATH '/autohome/ /MOCK_DATA.txt' INTO table local_table;
Loading data to table default.local_table
Table default.local_table stats: [numFiles=1, totalSize=96]
OK
Time taken: 0.787 seconds
hive>
    > select * from local_table;
OK
NULL    first_name
1       Sheffy
2       Luis
3       Evan
4       Page
5       Arda
6       Helen
7       Bambie
8       Kassie
9       Georgine
10      Doti
Time taken: 0.255 seconds, Fetched: 11 row(s)
```

Below steps show how to load the data from hdfs to hive:

```
[ @hadoop1 ~]$ hdfs dfs -ls /  /input/
Found 1 items
-rw-r--r--   3  hadoop         96 2018-04-24 12:35 /  /input/MOCK_DATA.txt
```

```
hive>
    > CREATE TABLE hdfs_table(id INT, name STRING)
    > Row format delimited
    > Fields terminated by '\t'
    > LOCATION '/  /input/';
OK
Time taken: 0.322 seconds
```

```
hive> select * from hdfs_table;
OK
NULL    first_name
1       Sheffy
2       Luis
3       Evan
4       Page
5       Arda
6       Helen
7       Bambie
8       Kassie
9       Georgine
10      Doti
Time taken: 0.789 seconds, Fetched: 11 row(s)
```

Data can be accessed via a simple query language and Hive supports overwriting or appending data.

Reference: https://www.tutorialspoint.com/hive/hive_create_table.htm

Login Procedure to Spark Cluster:

1. Login with your user name (CSX user name) to csh.cs.okstate.edu

2. MAC/LINUX users type ssh user@csh.cs.okstate.edu from your terminal. Windows users can use Putty to login to CSH.

3. Enter your CSX password when asked.

4. Then login into cshvm27 by typing ssh user@cshvm27

5. Enter your CSX password when asked.

There are 2 Spark versions available in the cluster

1. 2.0.1
2. 2.3.0

Up on deciding which version of spark set the path of spark version as follows:

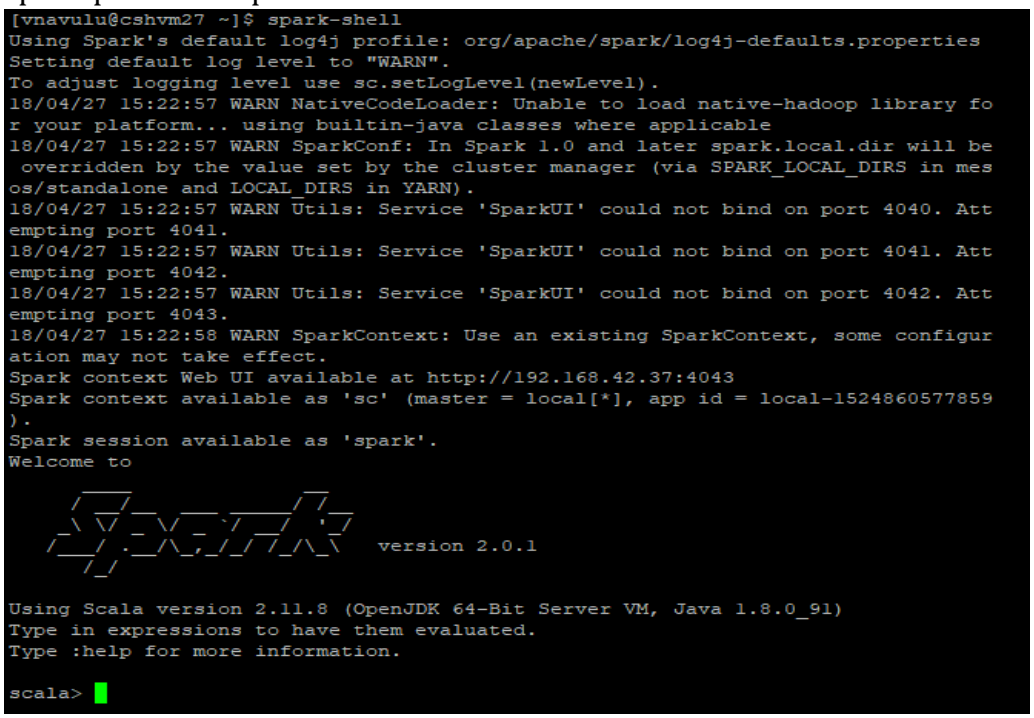
export SPARK_HOME=/root/spark-2.3.0

export PATH=$PATH:$SPARK_HOME/sbin:$SPARK_HOME/bin

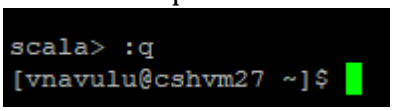
Once logging into spark cluster, Spark's API can be used through interactive shell or using programs written in Java, Scala and Python.

Steps to invoke Spark Shell:

1. After logging into spark cluster and following the steps mentioned above, type **spark-shell** at command prompt to start Spark's interactive shell. It will start the shell as shown below.



2. Give command :q or press Ctrl+D to exit the spark shell.



NOTE: Only commands written in scala would work in **spark-shell**. To use commands written in python, type **pyspark** to start spark shell. Use exit() command to exit from pyspark.

Steps to Run a spark Application written in Scala:

Once the code for the application is written, the application needs to be built using sbt.

1. Create a dependency file called build.sbt or build.xml file.
2. An application which have import statements such as sample file will have following contents:

```
name := "SparkAssignment"

version := "1.0"

scalaVersion := "2.10.4"

libraryDependencies ++=Seq(
  "org.apache.spark" %% "spark-core" % "2.0.1" % "provided"
  "org.apache.spark" %% "spark-sql" % "2.0.1",
  "org.apache.spark" % "spark-graphx_2.10" % "2.1.0",
  "org.apache.spark" % "spark-streaming_2.10" % "2.1.0"
)
```

3. Name should include name of the application.
4. Version is sbt version
5. Scala Version is the version of scala version present in the cluster.
6. libraryDependencies include list of libraries used while developing the spark application. Format is as follows:

```
libraryDependencies ++=Seq(
        "org.apache.spark" %% "<Spark Library Name>" % “<Spark Version> %
"provided"
)
```

7. Every spark application written in scala should have following dependencies imported:
    a. import org.apache.spark.{SparkConf, SparkContext}
    b. import org.apache.spark.SparkContext._
8. SparkConf is for creating the spark configuration. SparkContext is for creating spark context object which is used to run spark application
9. Hence build.sbt file will have atleast one entry in the library dependencies:

```
libraryDependencies ++=Seq(
  "org.apache.spark" %% "spark-core" % "2.0.1" % "provided")
```

10. Once build.sbt file is created, create folder structure as below in the folder where build.sbt is saved:
    a. /src/main/scala
11. Copy the spark application to /src/main/scala folder.
12. Go the folder where build.sbt is present.
13. Run following commands to build application using all included dependencies.
    a. sbt clean

```
[vnavulu@cshvm27 PA4]$ sbt clean
[info] Set current project to Spark Application (in build file:/oldhome/vnavulu/PA4/)
[success] Total time: 0 s, completed May 4, 2018 12:43:20 PM
[vnavulu@cshvm27 PA4]$
```

b. sbt package

```
[vnavulu@cshvm27 PA4]$ sbt package
[info] Set current project to Spark Application (in build file:/oldhome/vnavulu/PA4/)
[info] Updating {file:/oldhome/vnavulu/PA4/}pa4...
[info] Resolving org.scala-lang#scala-reflect;2.10.4 ...
[info] Done updating.
[info] Compiling 3 Scala sources to /oldhome/vnavulu/PA4/target/scala-2.10/classes...
[info] Packaging /oldhome/vnavulu/PA4/target/scala-2.10/spark-application_2.10-1.0.jar ...
[info] Done packaging.
[success] Total time: 25 s, completed May 4, 2018 12:44:43 PM
```

This command would create folder structure as shown below:

```
[vnavulu@cshvm27 PA4]$ ll
total 4
-rw-rw-r-- 1 vnavulu vnavulu 165 May  4 12:43 build.sbt
drwxrwxr-x 3 vnavulu vnavulu  19 Apr  8  2017 project
drwxrwxr-x 3 vnavulu vnavulu  17 Apr  8  2017 src
drwxrwxr-x 5 vnavulu vnavulu  60 May  4 12:44 target
```

c. project folder will contain class files

d. target folder would contain jar file created

```
[vnavulu@cshvm27 scala-2.10]$ pwd
/oldhome/vnavulu/PA4/target/scala-2.10
[vnavulu@cshvm27 scala-2.10]$ ll
total 36
drwxrwxr-x 2 vnavulu vnavulu  4096 May  4 12:44 classes
-rw-rw-r-- 1 vnavulu vnavulu 30268 May  4 12:44 spark-application_2.10-1.0.jar
```

14. Once the build is successful, then application can be run using following command:
    Ex: spark-submit - -class <class Name> <path to jar file>

```
spark-submit --class wordCount ./target/scala-2.10/spark-application_2.10-1.0.jar
```

NOTE: Before running the application is cluster, it is advisable to run the application as series of commands in spark-shell to test if the commands are working as intended or not.

Implementing Map Reduce in Spark using Scala:

Scala has map() and reduce() function that can be applied on RDD, but cannot be used directly to run map reduce paradigm of hadoop.

An example word count program is explained below to understand how to write a map reduce program in Scala.

```
import org.apache.spark.{SparkConf, SparkContext}
object wordCount {
  def main(args:Array[String]){
    val conf=new SparkConf().setAppName("Word Count Demo")
    val sc=new SparkContext(conf)
    val inputpath="WordCount.txt"
    // read in text file and split each document into words
    val textFile = sc.textFile(inputpath)
    val counts = textFile.flatMap(line => line.split(" "))
      .map(word => (word, 1))
      .reduceByKey(_ + _)
    counts.foreach(println)
  }
}
```

1. Import both SparkConf and SparkContext libraries.
2. Create a wordcount object which is like class in Java.
   Object wordCound{ }
3. Inside object write main method.
   def main(args:Array[String]) { }
4. Create SparkConf() object and set the name of of application in setAppName
   `val conf=new SparkConf().setAppName("Word Count Demo")`
5. Create Spark context object by passing the parameter as conf which is saved before.
6. Provide the location of input path.
7. Read the file as text file
8. Split each line using delimiter and store in line. (Here d
   `line => line.split(" ")`
9. Apply flatMap() transformation to get an iterator for the words.
10. Create an RDD pair with key as word and value as 1 for all the words using following transformation.
    `map(word => (word, 1))`
11. Combine the values with same key to generate a key, value pair where key is the word and value is count of words in the given file.
    `reduceByKey(_ + _)`
12. Print the output using following command.

```
counts.foreach(println)
```

13. Save the program as wordCount.scala. Make sure that object name and file name should be same.
14. Follow the steps given in the above section and run the program.
15. Sample output is as follows: