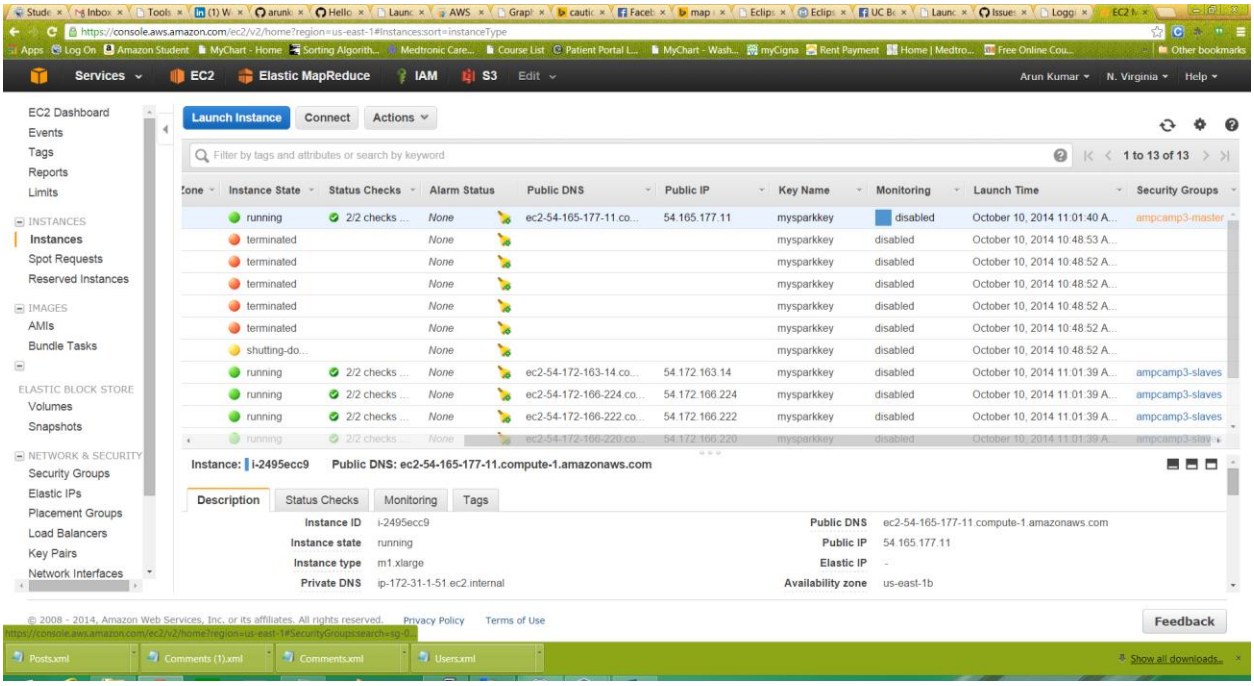
Hands-on Exercises – AMPCAMP 4

Note: Prerequisite – Cluster is running fine with data loaded.

These exercises are divided into sections designed to give a hands-on experience with various software components of the Berkeley Data Analytics Stack (BDAS). For Spark, we will walk you through using the Spark shell for interactive exploration of data. You have the choice of doing the exercises using Scala or using Python. For Shark, you will be using SQL in the Shark console to interactively explore the same data. The advanced modules may use other data sets, such as Twitter data for Spark Streaming.

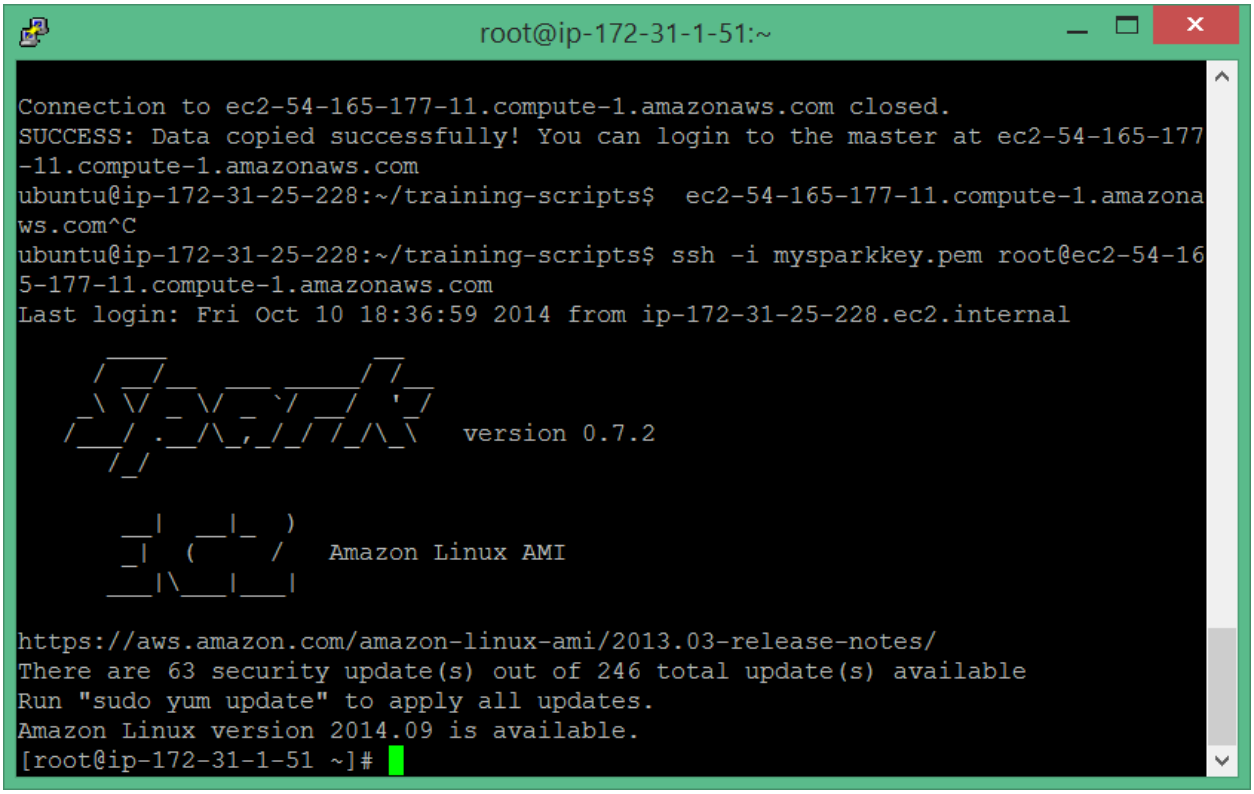
Link to exercise : http://ampcamp.berkeley.edu/4/exercises/logging-into-the-cluster.html

Get the public ip of the masternode from any one way described in the eariler document.



Login to master node (use your public IP).

ssh -i mysparkkey.pem root@ec2-54-165-177-11.compute-1.amazonaws.com

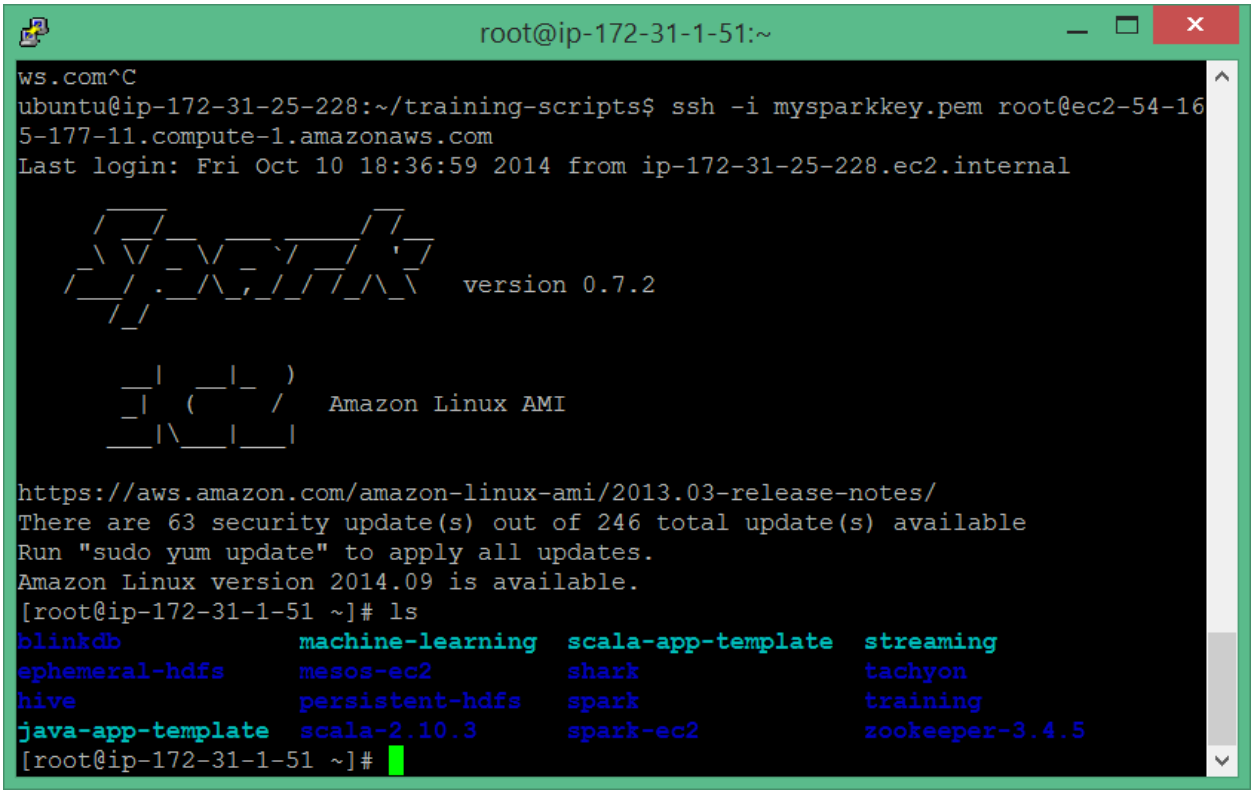


1. Cluster Details

Your cluster contains 6 m1.xlarge Amazon EC2 nodes. One of these 6 nodes is the master node, responsible for scheduling tasks as well as maintaining the HDFS metadata (a.k.a. HDFS name node). The other 5 are the slave nodes on which tasks are actually executed. You will mainly interact with the master node. If you haven't already, let's ssh onto the master node (see instructions above).

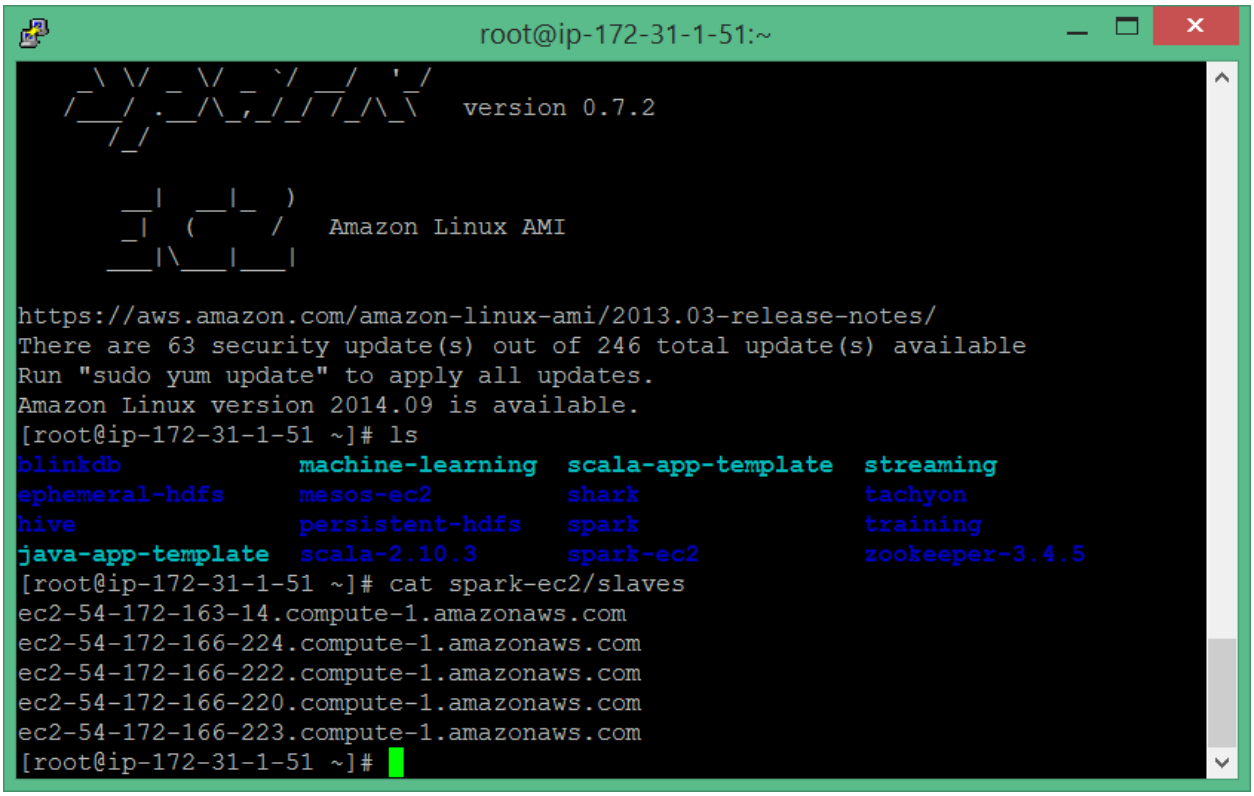
Once you've used SSH to log into the master, run the `ls` command and you will see a number of directories. Some of the more important ones are listed below:

ls



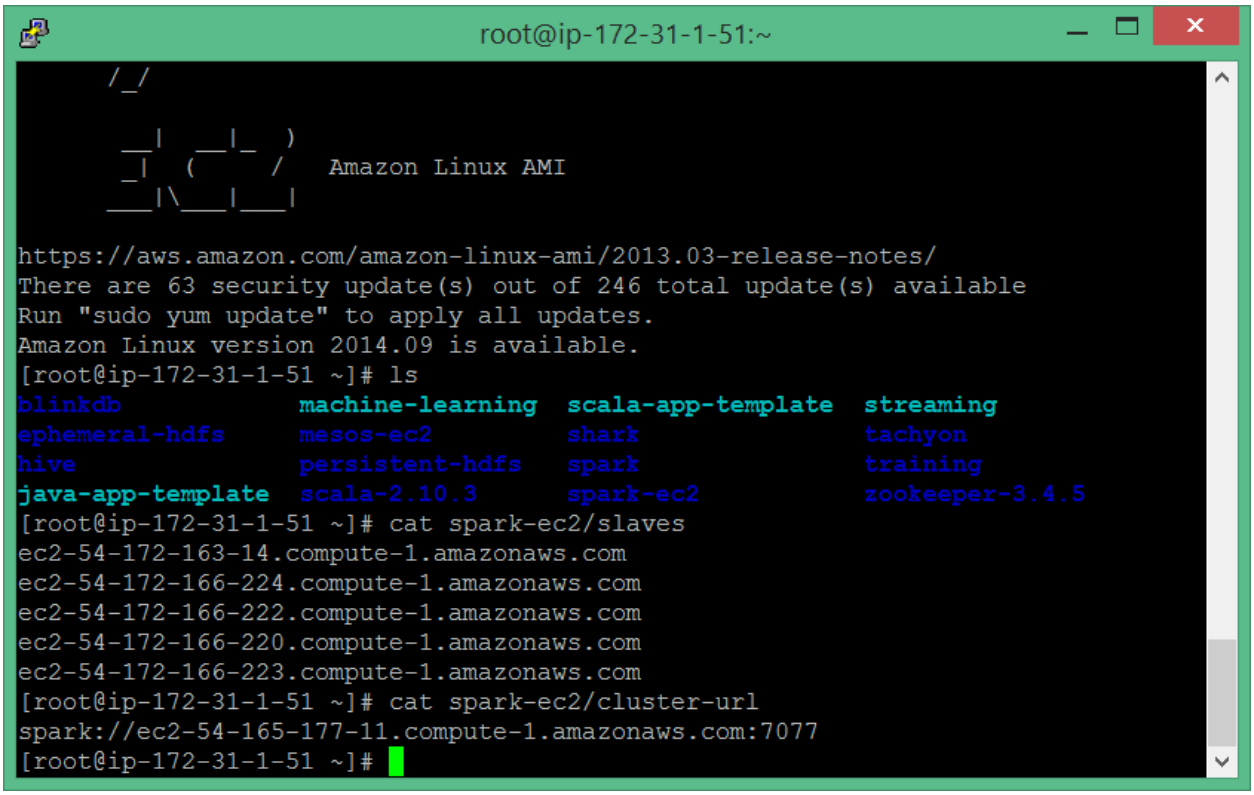
You can find a list of your 5 slave nodes in spark-ec2/slaves:

cat spark-ec2/slaves



For stand-alone Spark programs, you will have to know the Spark cluster URL. You can find that in spark-ec2/cluster-url:

cat spark-ec2/cluster-url

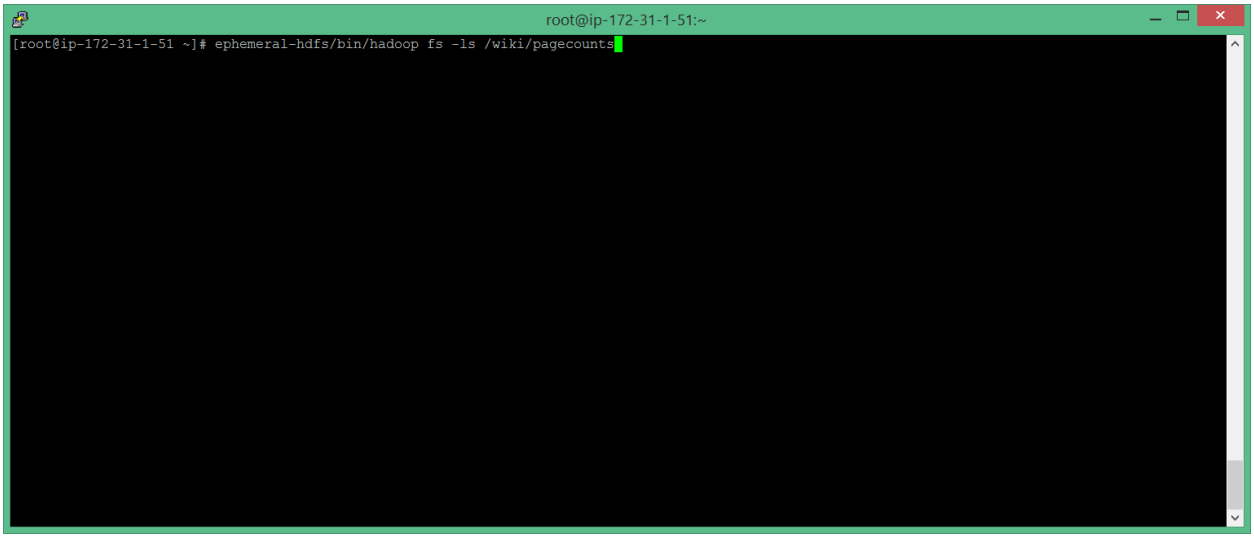


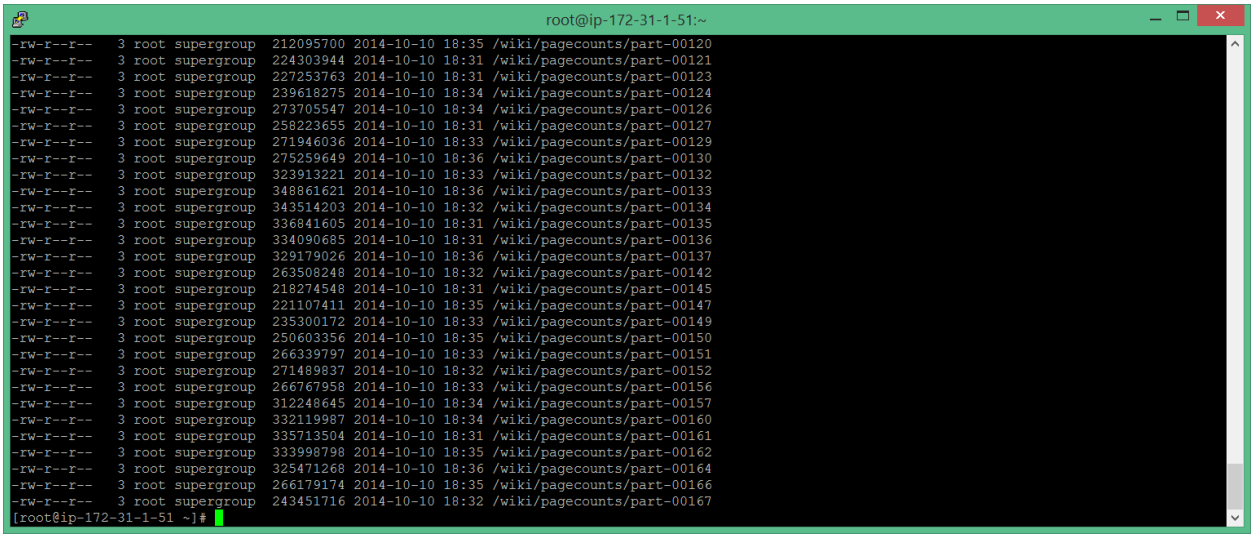
spark://ec2-54-165-177-11.compute-1.amazonaws.com:7077

2. Dataset For Exploration

Among other datasets, your HDFS cluster should come preloaded with 20GB of Wikipedia traffic statistics data obtained from http://aws.amazon.com/datasets/4182 . To make the analysis feasible (within the short timeframe of the exercise), we took three days worth of data (May 5 to May 7, 2009; roughly 20G and 329 million entries). You can list the files:

ephemeral-hdfs/bin/hadoop fs -ls /wiki/pagecounts

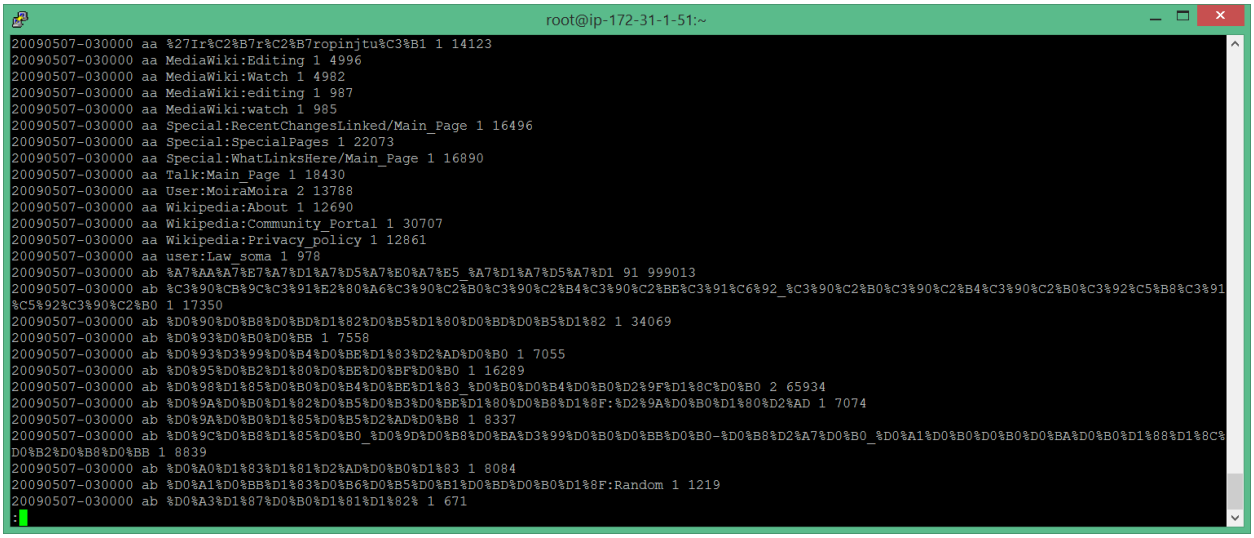




There are 74 files (2 of which are intentionally left empty). - Found 49 items

The data are partitioned by date and time. Each file contains traffic statistics for all pages in a specific hour. Let's take a look at the file:

```
ephemeral-hdfs/bin/hadoop fs -cat /wiki/pagecounts/part-00147 | less
```



The first few lines of the file are copied here:

20090507-030000 aa %27Ir%C2%B7r%C2%B7ropinjtu%C3%B1 1 14123

20090507-030000 aa MediaWiki:Editing 1 4996

20090507-030000 aa MediaWiki:Watch 1 4982

20090507-030000 aa MediaWiki:editing 1 987

20090507-030000 aa MediaWiki:watch 1 985

20090507-030000 aa Special:RecentChangesLinked/Main_Page 1 16496

Each line, delimited by a space, contains stats for one page. The schema is:

<date_time> <project_code> <page_title> <num_hits> <page_size>

The <date_time> field specifies a date in the YYYYMMDD format (year, month, day) followed by a hyphen and then the hour in the HHmmSS format (hour, minute, second). There is no information in mmSS. The <project_code> field contains information about the language of the pages. For example, project code “en” indicates an English page. The <page_title> field gives the title of the Wikipedia

page. The <num_hits> field gives the number of page views in the hour-long time slot starting at <data_time>. The <page_size> field gives the size in bytes of the Wikipedia page.

To quit less, stop viewing the file, and return to the command line, press q.

Introduction to SCALA

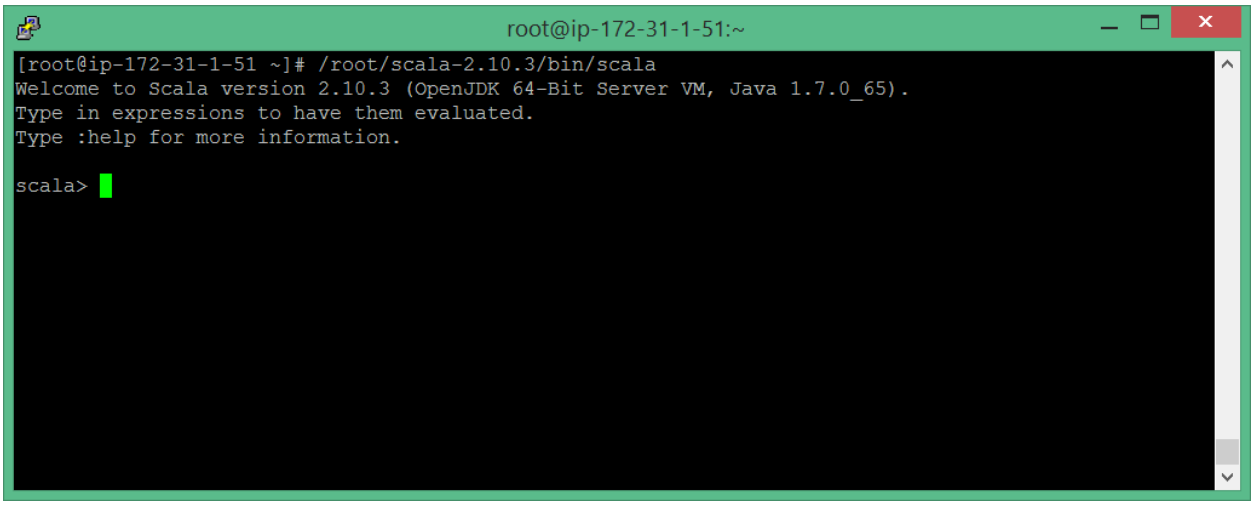
http://ampcamp.berkeley.edu/4/exercises/introduction-to-the-scala-shell.html

This chapter will teach you the basics of using the Scala shell and introduce you to functional programming with collections.

http://www.artima.com/scalazine/articles/steps.html

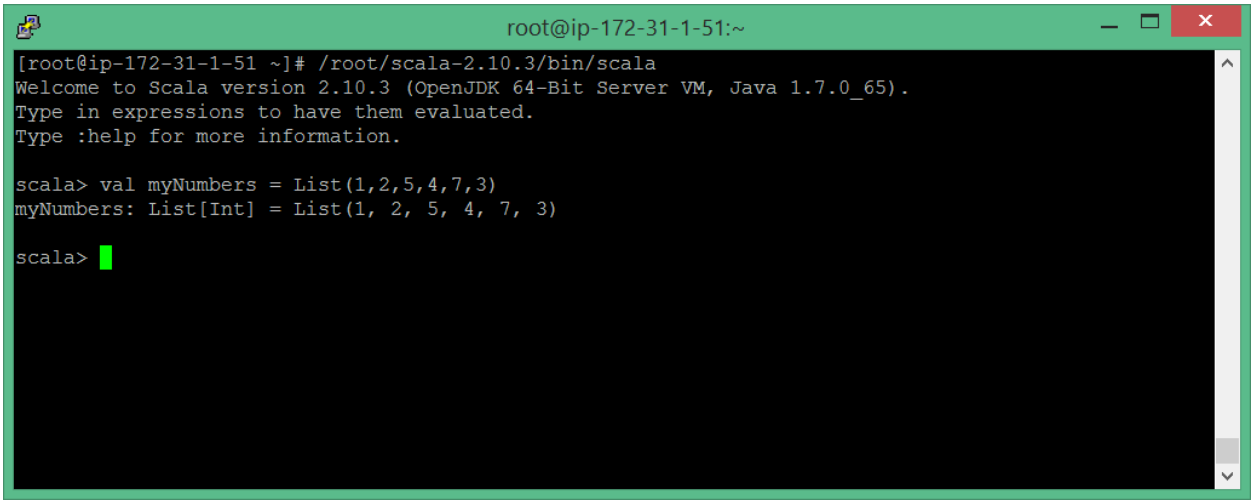
1. Launch the Scala console by typing:

```
/root/scala-2.10.3/bin/scala
```



2. Declare a list of integers as a variable called "myNumbers".

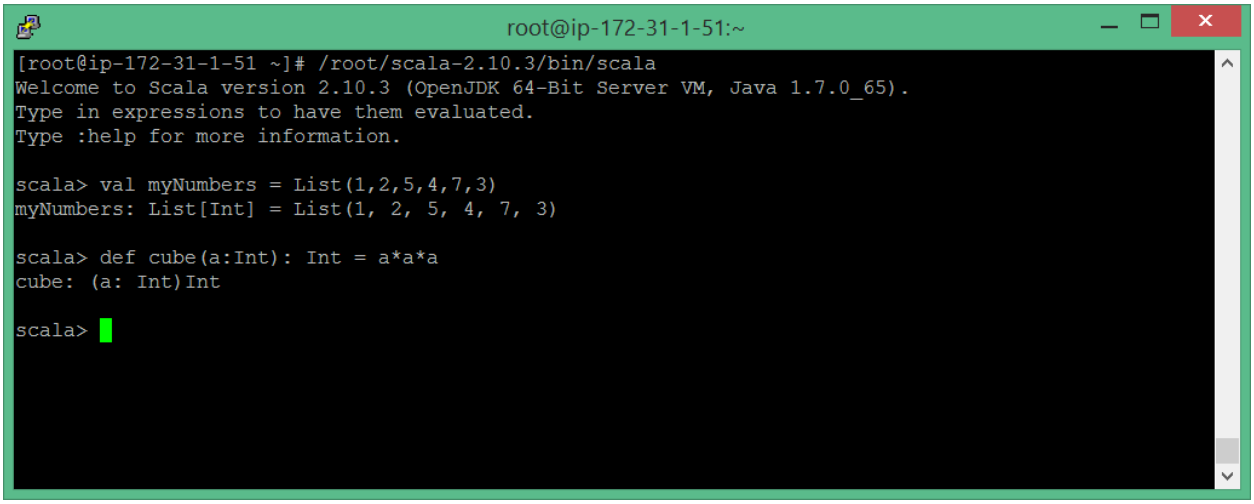
```
val myNumbers = List(1, 2, 5, 4, 7, 3)
```



Declare a function, cube, that computes the cube (third power) of an Int. See steps 2-4 of First Steps to Scala.

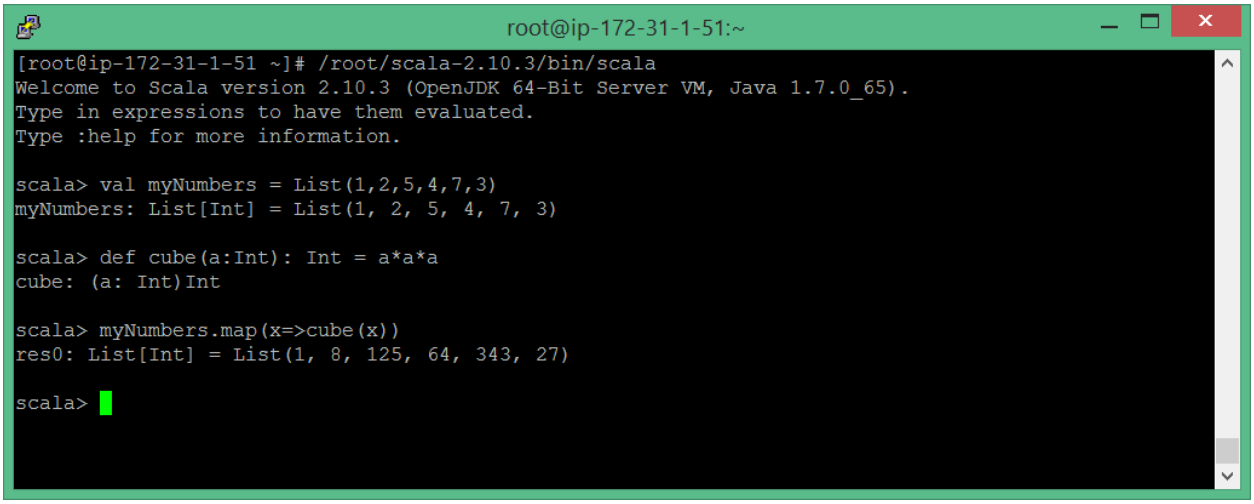
```
def cube(a: Int): Int = a * a * a

cube: (a: Int)Int
```



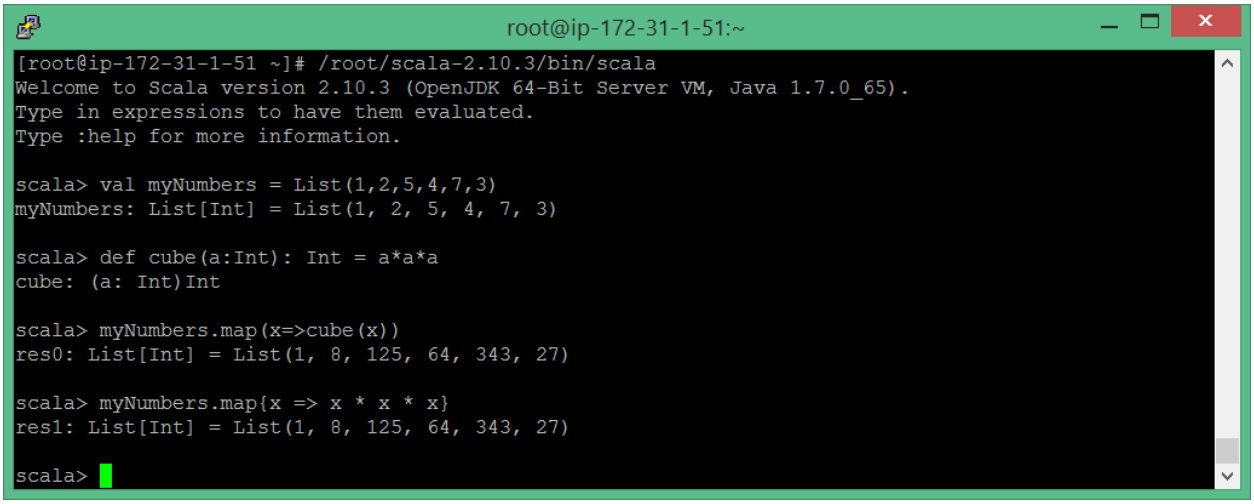
Apply the function to myNumbers using the map function. Hint: read about the map function in the Scala List API and also in Table 1 about halfway through the First Steps to Scala tutorial.

```
myNumbers.map(x => cube(x))
```



5. Then also try writing the function inline in a map call, using closure notation.

```
myNumbers.map{x => x * x * x}
```

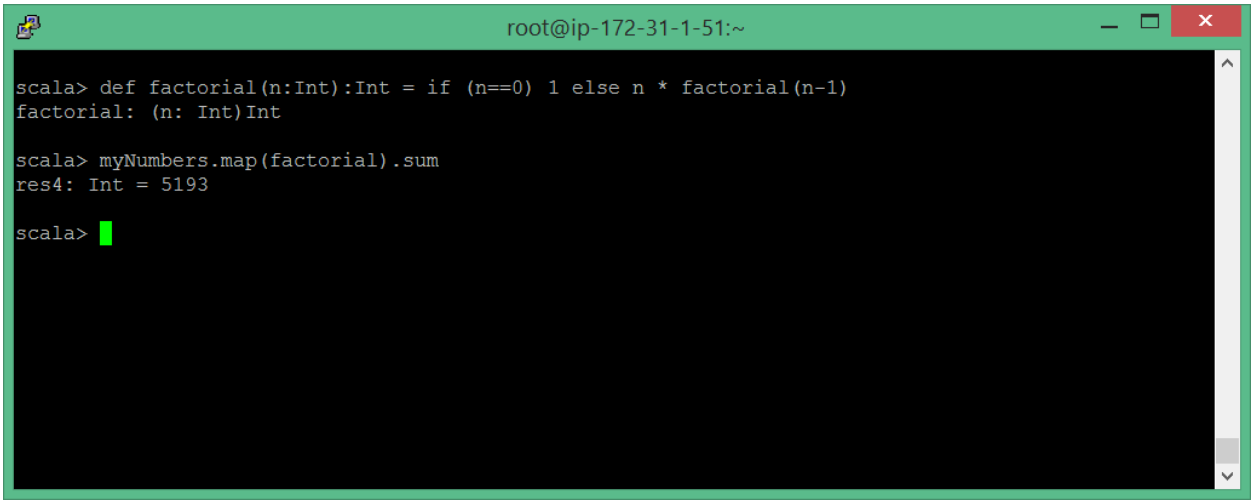


Define a factorial function that computes n! = 1 * 2 * … * n given input n. You can use either a loop or recursion, in our solution we use recursion (see steps 5-7 of First Steps to Scala). Then compute the sum of factorials in myNumbers. Hint: check out the sum function in the Scala List API.

```
def factorial(n:Int):Int = if (n==0) 1 else n * factorial(n-1)

myNumbers.map(factorial).sum
```

Note : To clear screen use 'CTRL + L'

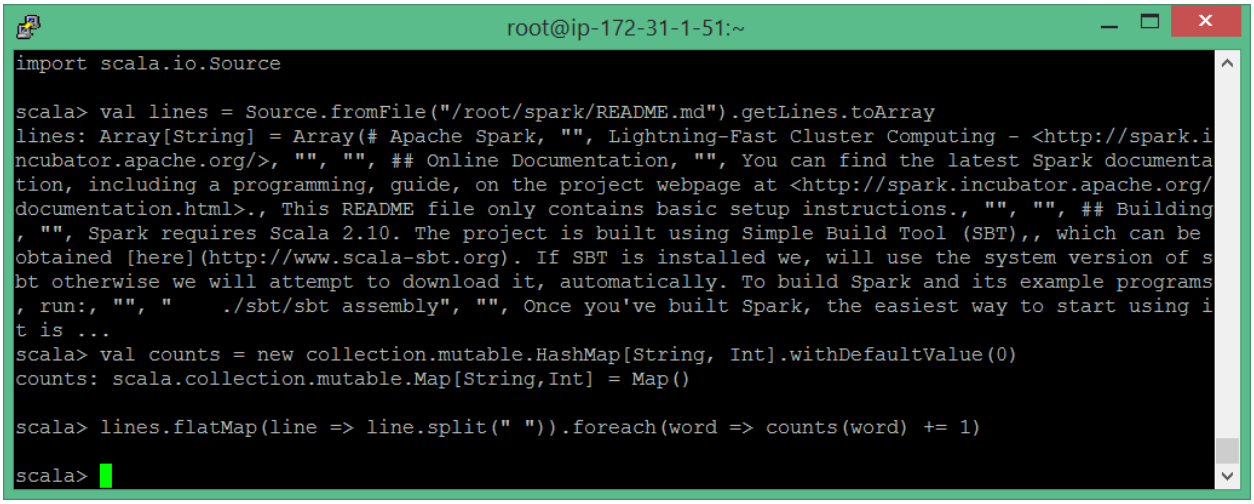


7. BONUS QUESTION. This is a more challenging task and may require 10 minutes or more to complete

```
1. import scala.io.Source
2. val lines = Source.fromFile("/root/spark/README.md").getLines.toArray
3. val lines = Source.fromFile("/root/spark/README.md").getLines.toArray
```


```
val counts = new collection.mutable.HashMap[String, Int].withDefaultValue(0)
```


```
lines.flatMap(line => line.split(" ")).foreach(word => counts(word) += 1)
```



counts