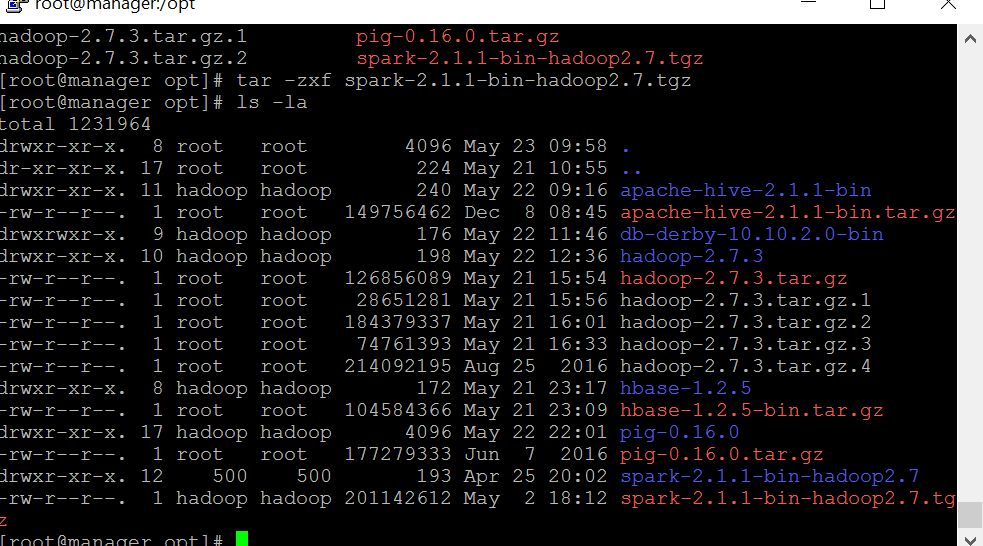
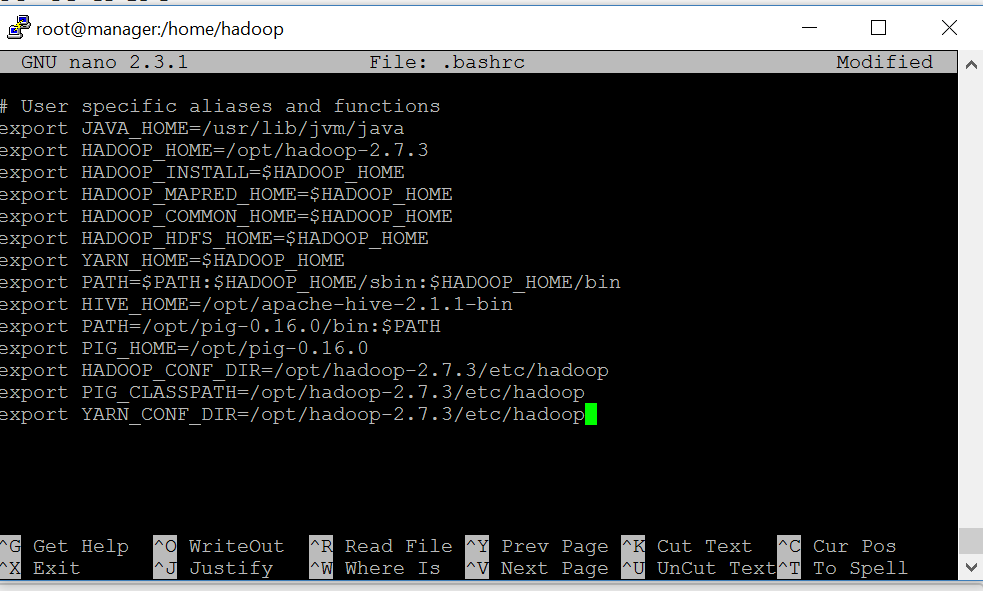
**Apache Spark Lab:**

Downloaded Apache Spark distribution:

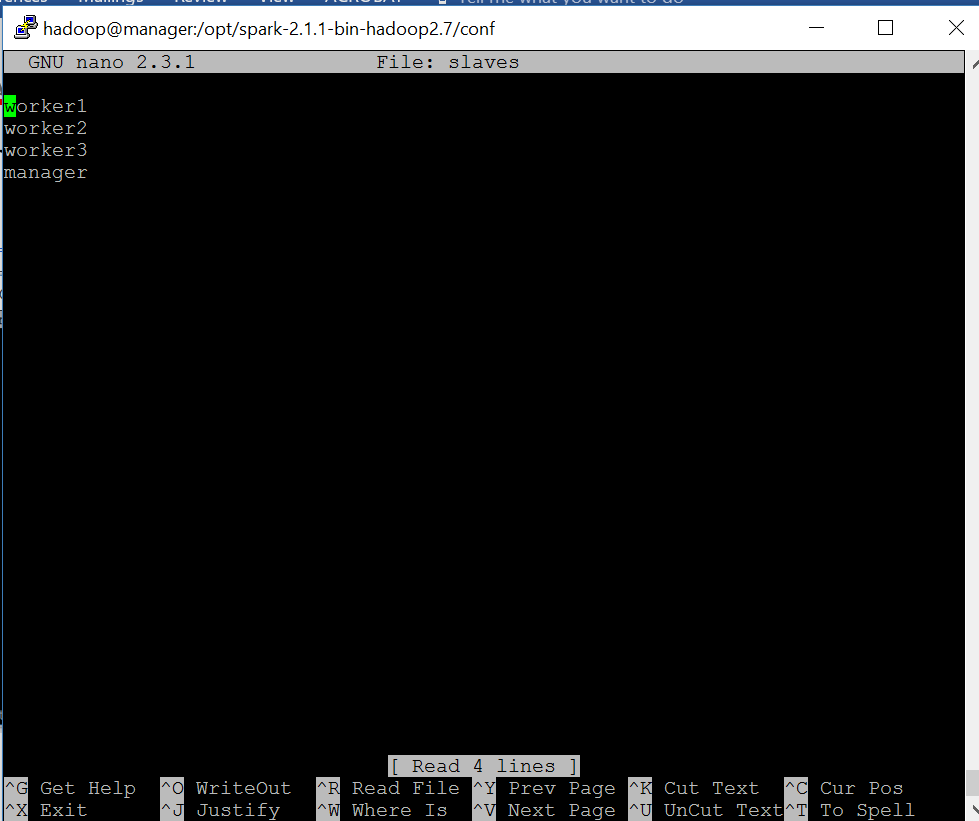
Direct download url: https://d3kbcqa49mib13.cloudfront.net/spark-2.1.1-bin-hadoop2.7.tgz



Edited .bashrc file to include HADOOP\_CONF\_DIR AND YARN\_CONF\_DIR properties



Edited slaves file with worker nodes



Secure copied Apache Spark base directory to worker1,worker2,worker3.

Changed owner to Hadoop for base directory across all slave nodes.

Before we start spark jobs we need to make sure hdfs and yarn are started.

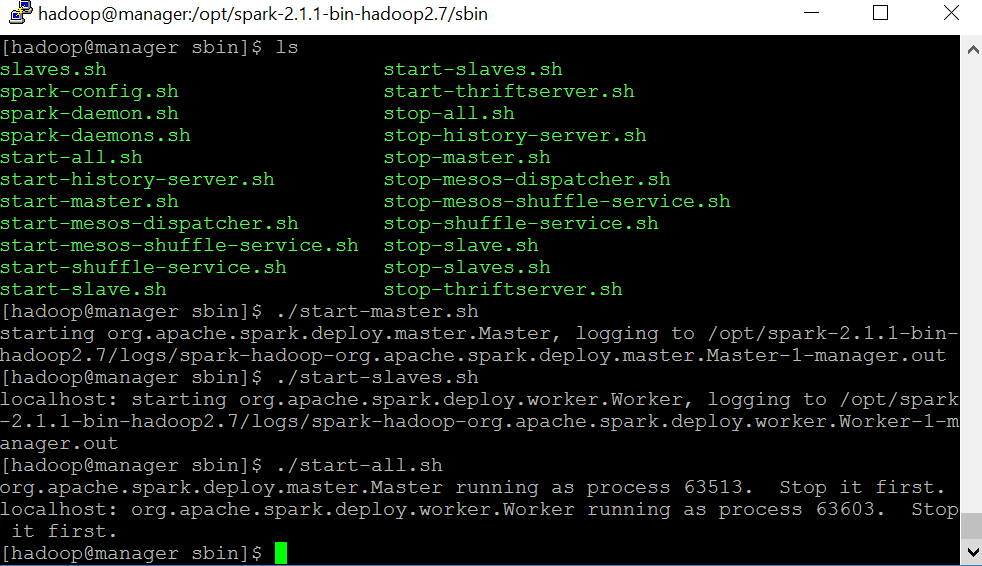
Steps to start pyspark console and dependencies:

1.Started dfs.sh

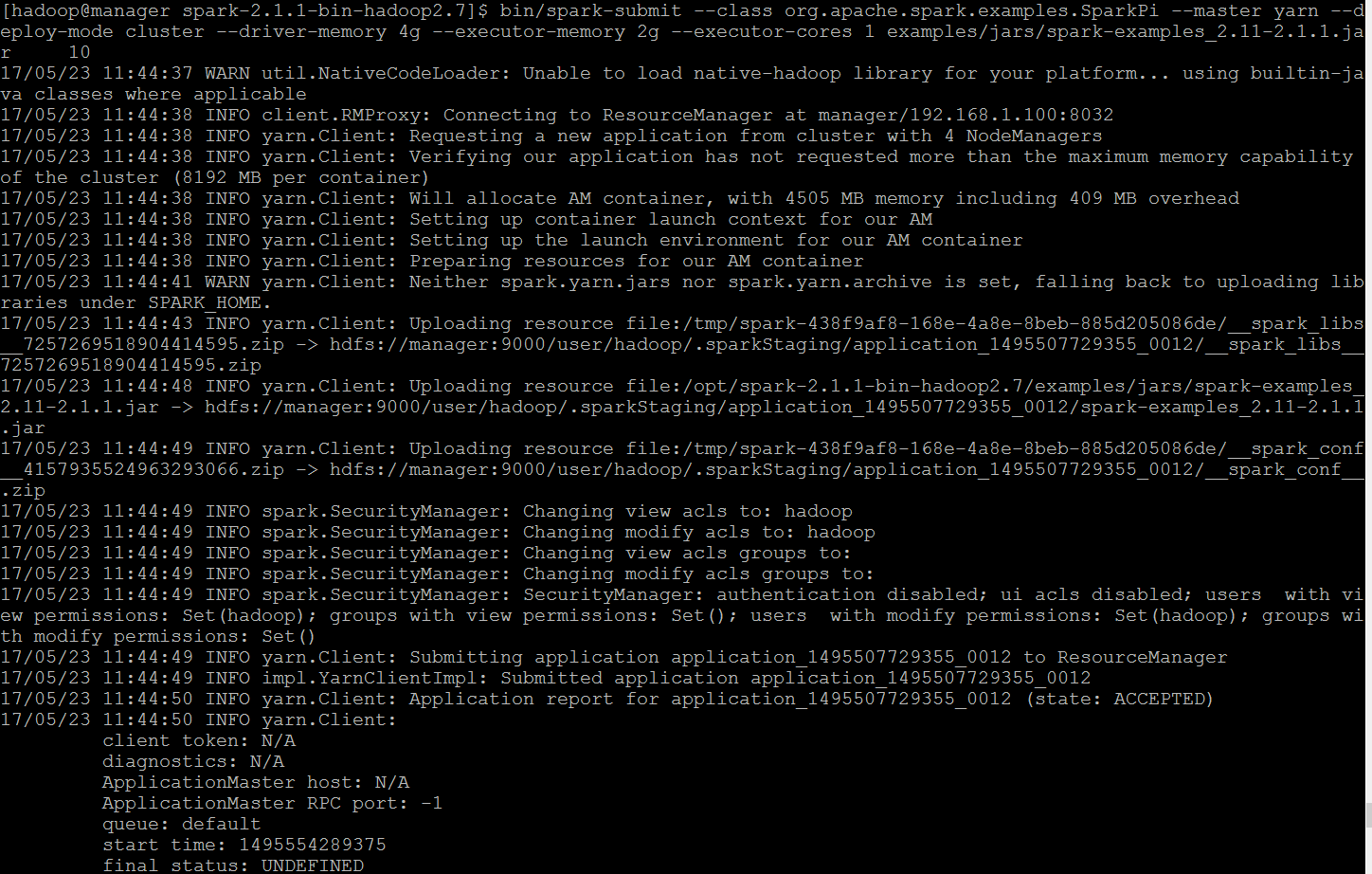
2.Started yarn .sh

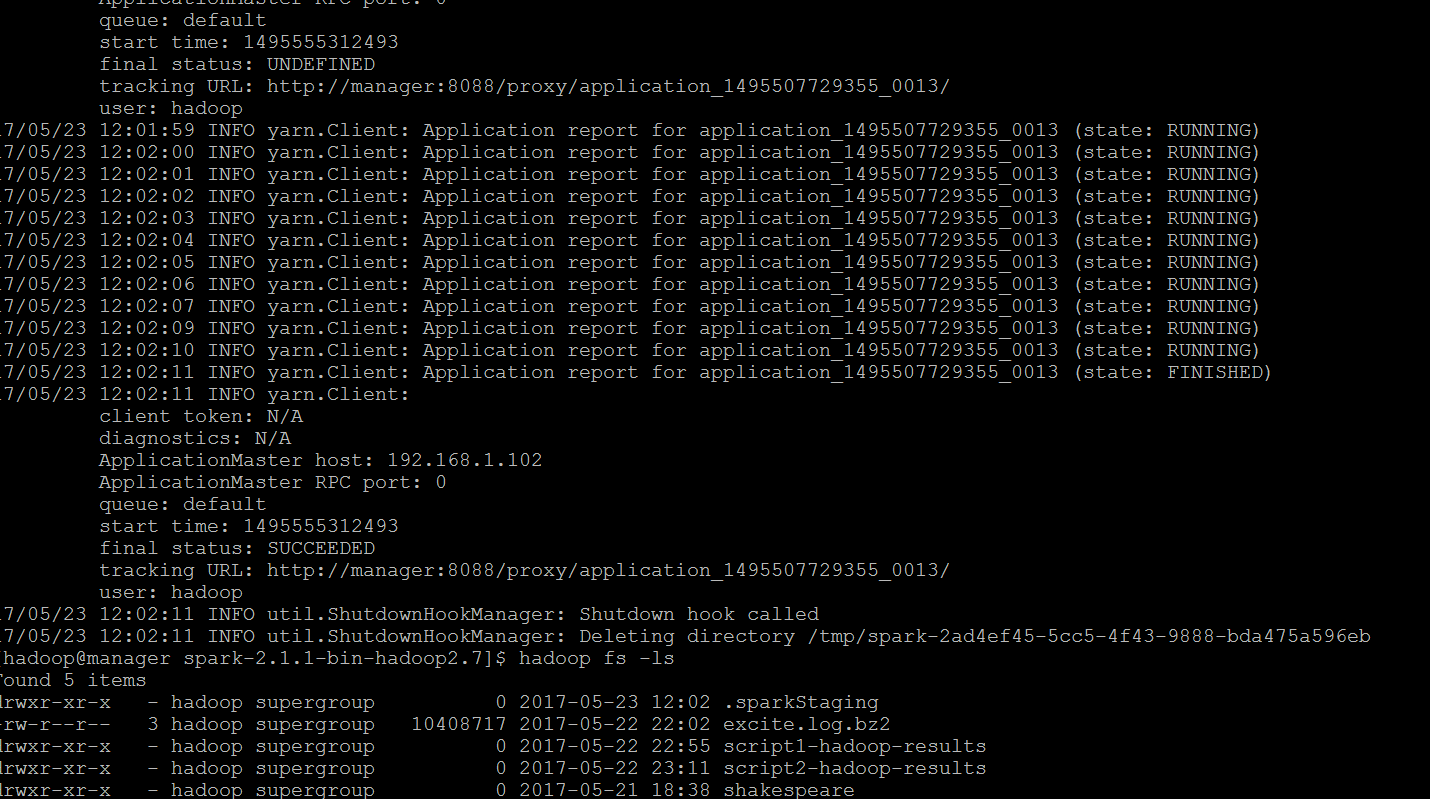
3.Start-all

4.Bin/pyspark



Submitting SparkPi job to cluster



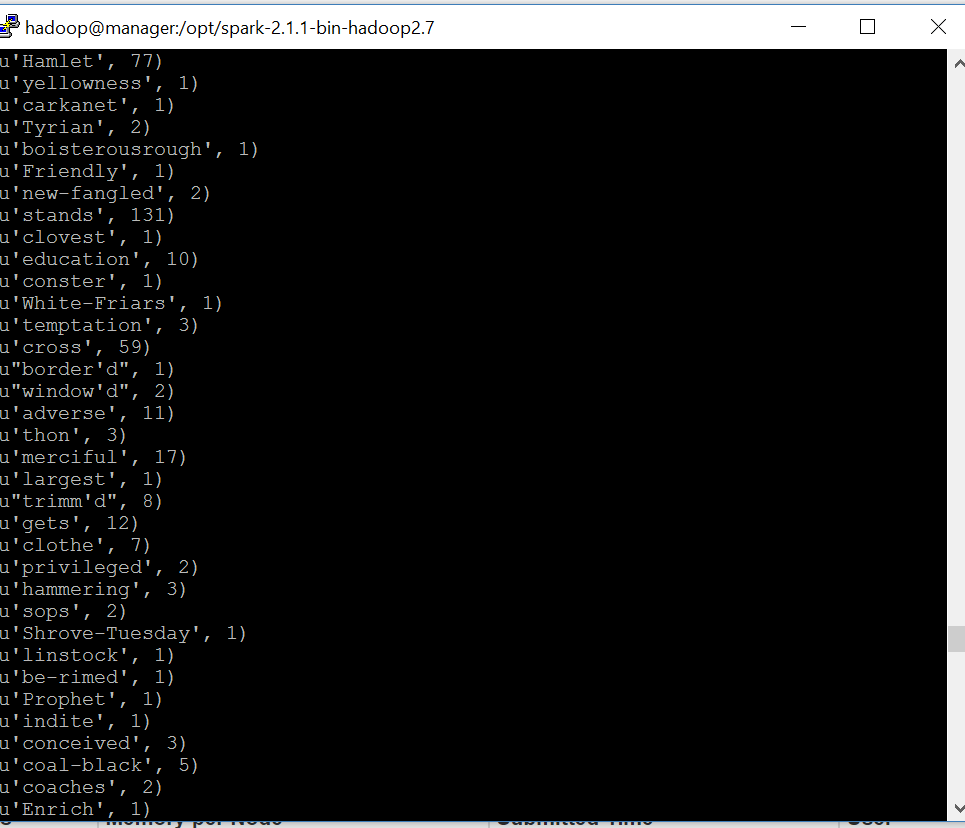


Spark Java and Scala Examples:

Example I. Shakespeare wordcount Map Reduce Example

The following output is the result of a custom wordcount function on Shakespeare text on HDFS.

Explanation: First we source the Shakespeare file into a in memory data structure. Then we define a lambda function. We map the function and reduce it to obtain a list of words in the Shakespeare text organized by word counts.



Example II. Keyword Search example in log files

textFile = sc.textFile("hdfs://...")

*# Creates a DataFrame having a single column named "line"*

df = textFile.map(**lambda** r: Row(r)).toDF(["line"])

errors = df.filter(col("line").like("*%E*RROR%"))

*# Counts all the errors*

errors.count()

*# Counts errors mentioning MySQL*

errors.filter(col("line").like("%MySQL%")).count()

*# Fetches the MySQL errors as an array of strings*

errors.filter(col("line").like("%MySQL%")).collect()

Explanation: This snippet of code loads a text log file in memory and then scans the file for occurences of a keyword(“error”). We can returna count of all keywords using this method to count the number of errors we had for our application.

Example II.Spark Pi example

List<Integer> l = **new** ArrayList<>(NUM\_SAMPLES);

**for** (int i = 0; i < NUM\_SAMPLES; i++) {

l.add(i);

}

long count = sc.parallelize(l).filter(i -> {

double x = Math.random();

double y = Math.random();

**return** x\*x + y\*y < 1;

}).count();

System.out.println("Pi is roughly " + 4.0 \* count / NUM\_SAMPLES);

Explanation: In this example we estimate the value of pi by calculating random points across the unit square(0,0) till (1,1). The value is derived using pi/4

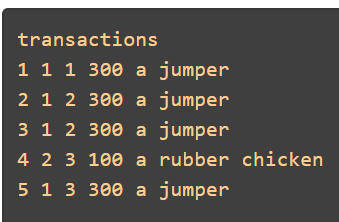
Example III. User and Transaction GeoLocation example in Java

I have two datasets:

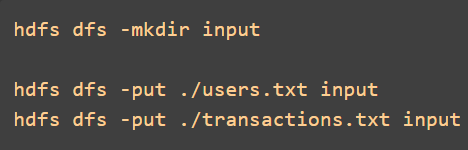
1. User information (id, email, language, location)
2. Transaction information (transaction-id, product-id, user-id, purchase-amount, item-description)

Given these datasets, I want to find the number of unique locations in which each product has been sold. To do that, I need to join the two datasets together





First we create hdfs entries for the text files containing these data sets.



The main method invokes a chain of predefined functions find the number of unique locations in which each product has been sold. The steps that happen are as follows.

1. read / transform transactions data
2. read / transform users data
3. left outer join of transactions on users
4. get rid of user\_id key from the result of the previous step by applying values()
5. find distinct() values
6. countByKey()
7. transform result to an RDD
8. save result to Hadoop

Implementation Code:



