Apache Hadoop and Spark Fundamentals (Third Edition)
You can find more information and download the sandbox at:
http://hortonworks.com/products/hortonworks-sandbox/

Apache Hadoop and Spark Fundamentals (Third Edition)
LESSON 2.2 INSTALL APACHE HADOOP, PIG, AND HIVE ON LAPTOP OR DESKTOP
Step 1: Download Hadoop
==========
Unless otherwise noted the following steps are done by root
Download the latest distribution from the Hadoop web site (http://hadoop.apache.org/).

```
  wget -P /tmp http://mirrors.ibiblio.org/apache/hadoop/common/hadoop-2.8.1/
hadoop-2.8.1.tar.gz
```

Next extract the package in /opt:

```
  mkdir -p /opt/
  tar xvzf /tmp/hadoop-2.8.1.tar.gz -C /opt
```

Step 2: Set JAVA_HOME and HADOOP_HOME
==========
version 8 (openjdk 1.8) will be installed:

```
  yum install java-1.8.0-openjdk java-1.8.0-openjdk-devel
```

to check which version of java is default run "java -version"
Add java.sh to profile.d

```
echo 'export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk.x86_64'> /etc/profile.d/java.sh
```

Add hadoop.sh to profile.d

```
echo 'export HADOOP_HOME=/opt/hadoop-2.8.1;export PATH=$HADOOP_HOME/bin:$PATH'
>/etc/profile.d/hadoop.sh
```

To make sure JAVA_HOME and HADOOP_HOME are defined for this session, source the new
script:

```
 source /etc/profile.d/java.sh
 source /etc/profile.d/hadoop.sh
```

Step 3: Create Users and Groups
==========
```
  groupadd hadoop
  useradd -g hadoop yarn
  useradd -g hadoop hdfs
  useradd -g hadoop mapred
```

Step 4: Make Data and Log Directories
==========
```
  mkdir -p /var/data/hadoop/hdfs/nn
  mkdir -p /var/data/hadoop/hdfs/snn
  mkdir -p /var/data/hadoop/hdfs/dn
```

```
  chown hdfs:hadoop /var/data/hadoop/hdfs -R

#  Create the log directory and set the owner and group as follows:
  cd /opt/hadoop-2.8.1
  mkdir logs
  chmod g+w logs
  chown -R yarn:hadoop .

Step 5: Configure core-site.xml
==============================
# Add the following properties to /opt/hadoop-2.8.1/etc/hadoop/core-site.xml
<configuration>
      <property>
            <name>fs.default.name</name>
            <value>hdfs://localhost:9000</value>
      </property>
      <property>
            <name>hadoop.http.staticuser.user</name>
            <value>hdfs</value>
      </property>
</configuration>

Step 6: Configure hdfs-site.xml
==============================
# Add the following properties to /opt/hadoop-2.8.1/etc/hadoop/hdfs-site.xml
<configuration>
 <property>
   <name>dfs.replication</name>
   <value>1</value>
 </property>
 <property>
   <name>dfs.namenode.name.dir</name>
   <value>file:/var/data/hadoop/hdfs/nn</value>
 </property>
 <property>
   <name>fs.checkpoint.dir</name>
   <value>file:/var/data/hadoop/hdfs/snn</value>
 </property>
 <property>
   <name>fs.checkpoint.edits.dir</name>
   <value>file:/var/data/hadoop/hdfs/snn</value>
 </property>
 <property>
   <name>dfs.datanode.data.dir</name>
   <value>file:/var/data/hadoop/hdfs/dn</value>
 </property>
</configuration>

Step 7: Configure mapred-site.xml
==============================
# copy the template file
  cp mapred-site.xml.template mapred-site.xml

# Add the following properties to /opt/hadoop-2.8.1/etc/hadoop/mapred-site.xml
```

```
<configuration>
<property>
  <name>mapreduce.framework.name</name>
  <value>yarn</value>
</property>
<property>
  <name>mapreduce.jobhistory.intermediate-done-dir</name>
  <value>/mr-history/tmp </value>
</property>
<property>
  <name> mapreduce.jobhistory.done-dir</name>
  <value>/mr-history/done</value>
</property>
</configuration>
```

Step 8: Configure yarn-site.xml

```
==============================
# Add the following properties to /opt/hadoop-2.8.1/etc/hadoop/yarn-site.xml
<configuration>
<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
 </property>
 <property>
  <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
  <value>org.apache.hadoop.mapred.ShuffleHandler</value>
 </property>
</configuration>
```

Step 9: Modify Java Heap Sizes

```
==============================
# Edit /opt/hadoop-2.8.1/etc/hadoop/hadoop-env.sh file to reflect the following
# (Don't forget to remove the "#" at the beginning of the line.):
HADOOP_HEAPSIZE=500
HADOOP_NAMENODE_INIT_HEAPSIZE="500"

# Next, in the same directory, edit the mapred-env.sh to reflect the following:
HADOOP_JOB_HISTORYSERVER_HEAPSIZE=250

# Edit yarn-env.sh to reflect the following:
JAVA_HEAP_MAX=-Xmx500m

#The following will need to added to yarn-env.sh
YARN_HEAPSIZE=500

# Finally, edit hadoop-env.sh and add the following to the end:
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="$HADOOP_OPTS -Djava.library.path=$HADOOP_HOME/lib "
```

Step 10: Format HDFS

```
====================
# As user "hdfs"
 su - hdfs
```

```
 cd /opt/hadoop-2.8.1/bin
 ./hdfs namenode -format

Step 11: Start the HDFS Services
==============================
# As user hdfs
  cd /opt/hadoop-2.8.1/sbin
  ./hadoop-daemon.sh start namenode
  ./hadoop-daemon.sh start secondarynamenode
  ./hadoop-daemon.sh start datanode

# If the daemon started, you should see responses above that will point to the log file.
# (Note that the actual log file is appended with ".log" not ".out.")
# Issue a jps command to see that all the services are running. The actual PID
# values will be different than shown in this listing:

$ jps

# All Hadoop services can be stopped using the hadoop-daemon.sh script.
# For example, to stop the datanode service enter the following
   ./hadoop-daemon.sh stop datanode

# create /mr-history for job history server directory in hdfs and make sure HDFS working
 hdfs dfs -mkdir -p /mr-history/tmp
 hdfs dfs -mkdir -p /mr-history/done
 hdfs dfs -chown -R yarn:hadoop  /mr-history
 hdfs dfs -chmod go+rwx /tmp

# There are two convenience scripts in the scripts directory to start and stop HDFS services
(run as root)
# You can add them to /etc/rc.local file
  start-hdfs.sh
  stop-hdfs.sh

Step 12: Start YARN Services
============================
# as user "yarn"
  su - yarn
  cd /opt/hadoop-2.8.1/sbin
  ./yarn-daemon.sh start resourcemanager
  ./yarn-daemon.sh start nodemanager
  ./mr-jobhistory-daemon.sh start historyserver

  jps

# Similar to HDFS, the services can be stopped by issuing a stop argument to the daemon
script:
  ./yarn-daemon.sh stop nodemanager

# There are two convenience scripts in the scripts directory to start and stop YARN services
(run as root)
# You can add them to /etc/rc.local file
  start-yarn.sh
  stop-yarn.sh
```

Step 13: Verify the Running Services Using the Web Interface
==========
As user "hdfs" to see the HDFS web interface (other browsers can be used):
 firefox http://localhost:50070

To see the ResourceManager (YARN) web interface:
 firefox http://localhost:8088

Run a Sample MapReduce Examples
 export YARN_EXAMPLES=/opt/hadoop-2.8.1/share/hadoop/mapreduce/

To test your installation, run the sample "pi" application
 yarn jar $YARN_EXAMPLES/hadoop-mapreduce-examples-2.8.1.jar pi 8 100000

Install Apache Pig

As root, get sources
  wget -P /tmp http://mirrors.ibiblio.org/apache/pig/pig-0.17.0/pig-0.17.0.tar.gz

Next extract the package in /opt:
  mkdir -p /opt/
  tar xvzf /tmp/pig-0.17.0.tar.gz -C /opt

set environment
  echo 'export PATH=$PATH:/opt/pig-0.17.0/bin; export PIG_HOME=/opt/pig-0.17.0; export
PIG_CLASSPATH=/opt/hadoop-2.8.1/etc/hadoop' >/etc/profile.d/pig.sh

Create a Pig user and change ownership (do as root)
  useradd -g hadoop pig
  chown -R pig:hadoop /opt/pig-0.17.0

Pig is ready for use by users (they must re-login or "source /etc/profile.d/pig.sh")

Install Apache Hive

As root, get sources, extract, create /etc/profile.d/hive.sh
  wget -P /tmp http://mirrors.ibiblio.org/apache/hive/hive-2.3.2/apache-hive-2.3.2-bin.tar.gz

  tar xvzf /tmp/apache-hive-2.3.2-bin.tar.gz -C /opt

  echo 'export PATH=$PATH:/opt/apache-hive-2.3.2-bin/bin; export HIVE_HOME=/opt/apache-
hive-2.3.2-bin' >/etc/profile.d/hive.sh

make needed directories in HDFS
  su - hdfs -c "hdfs dfs -mkdir -p /user/hive/warehouse"
  su - hdfs -c "hdfs dfs -chmod g+w /user/hive/warehouse"

Copy hive-site.xml file in files directory
  cp files/hive-site.xml /opt/apache-hive-2.3.2-bin/conf

Or, create conf/hive-site.xml and add the following:

```
<configuration>
<property>
  <name>javax.jdo.option.ConnectionURL</name>
  <value>jdbc:derby://localhost:1527/metastore_db;create=true</value>
  <description>JDBC connect string for a JDBC metastore</description>
</property>
<property>
  <name>javax.jdo.option.ConnectionDriverName</name>
  <value>org.apache.derby.jdbc.ClientDriver</value>
  <description>Driver class name for a JDBC metastore</description>
</property>
</configuration>

# remove the extra log4j-slf4j library (included in Hadoop install)
  mv /opt/apache-hive-2.3.2-bin/lib/log4j-slf4j-impl-2.6.2.jar /opt/apache-hive-2.3.2-bin/lib/
log4j-slf4j-impl-2.6.2.jar.extra

# Create a Hive user and change ownership (do as root)
  useradd -g hadoop hive
  chown -R hive:hadoop /opt/apache-hive-2.3.2-bin

# Install Apache Derby
# Hive needs a "metastore" database for metadata. The default is Apache Derby
# install Apache Derby version 10.13.1.1
  wget -P /tmp http://mirrors.gigenet.com/apache//db/derby/db-derby-10.13.1.1/db-
derby-10.13.1.1-bin.tar.gz

  tar xvzf /tmp/db-derby-10.13.1.1-bin.tar.gz -C /opt

# set the Derby environment defines. Note, derby database will be in $DERBY_HOME/data,
change as needed.
  echo 'export DERBY_HOME=/opt/db-derby-10.13.1.1-bin; export PATH=$DERBY_HOME/bin:
$PATH; export DERBY_OPTS="-Dderby.system.home=$DERBY_HOME/data"'>/etc/profile.d/
derby.sh

# source these file to make sure $DERBY_HOME and $HIVE_HOME are defined
  source /etc/profile.d/derby.sh
  source /etc/profile.d/hive.sh

# copy these libraries to $HIVE_HOME
  cp $DERBY_HOME/lib/derbyclient.jar $HIVE_HOME/lib
  cp $DERBY_HOME/lib/derbytools.jar $HIVE_HOME/lib

# Start (and Stop) Derby (nohup will leave log file in the directory you run command)
  nohup startNetworkServer -h 0.0.0.0 &

# To stop use "stopNetworkServer"
# Change derby to hive user
  chown -R hive:hadoop /opt/db-derby-10.13.1.1-bin

# configure Hive schema
  schematool -initSchema -dbType derby
```

```
# There are two convenience scripts in the scripts directory to start and stop Derby (run as
root)
# You can add them to /etc/rc.local file
  start-derby.sh
  stop-derby.sh

# Start/Test Hive
# Make sure all Hadoop services are running.
# As user hdfs
  su - hdfs

# Enter "hive" at prompt. Output as follows. Ignore "which: no hbase" warning
$ hive

# Make sure hive is working (simple test) and quit.
hive> show tables;
hive> quit;

Apache Hadoop and Spark Fundamentals (Third Edition)
LESSON 2.3 INSTALL APACHE SPARK ON LAPTOP OR DESKTOP
Step 1: Download Spark
======================
# Unless otherwise noted the following steps are done by root
# Note Spark 1.6.3 is used because CentOS 6.x only supports Python 2.6.6
  wget -P /tmp http://mirrors.ibiblio.org/apache/spark/spark-1.6.3/spark-1.6.3-bin-
hadoop2.6.tgz

# Next extract the package in /opt:
  mkdir -p /opt/
  tar xvzf /tmp/spark-1.6.3-bin-hadoop2.6.tgz -C /opt

Step2: Set Spark Path and JAVA_HOME
===================================
# Spark runs on Java 8, however, the Hadoop version we installed does not support Java 8
# Some changes are needed. Install Java 1.8 (as root)

  yum install java-1.8.0-openjdk java-1.8.0-openjdk-devel

# now set JAVA_HOME to the new path
# Also note, the install of java-1.8 will set /etc/alternatives/java to /usr/lib/jvm/jre-1.8.0-
openjdk.x86_64/bin/java

# to check which version of java is default run "java -version"
# Set spark profile and path
echo 'export PATH=$PATH:/opt/spark-1.6.3-bin-hadoop2.6/bin; export SPARK_HOME=i/opt/
spark-1.6.3-bin-hadoop2.6' >/etc/profile.d/spark.sh

# Create a Spark user and change ownership (do as root)
  useradd -g hadoop spark
  chown -R spark:hadoop /opt/spark-1.6.3-bin-hadoop2.6

# Create /tmp/hive for Spark use
  mkdir /tmp/hive
  chmod ugo+rwx /tmp/hive
```

```
# To turn off all the INFO messages
Change:
  # Set everything to be logged to the console
  log4j.rootCategory=INFO, console
To:
  # Set everything to be logged to the console
  log4j.rootCategory=WARN, console
  #log4j.rootCategory=INFO, console

in  /opt/spark-1.6.3-bin-hadoop2.6/conf/log4j.properties

Test Spark Install
==================
# If you have not logged out run ". /etc/profile.d/spark.sh" before running the tests.
# this file will be used automatic on all subsequent logins.
# Run the pi example
  run-example SparkPi 10

# Start the Spark shell (in Scala) Use ":q" to quit. Starts a local version with one thread.
  spark-shell

# Start a Spark shell in Python (Python must be installed) Starts a local master with two threads
# using the "--master local[2]" option. Use ctrl-D or "quit()" to quit.
  pyspark --master local[2]

# also run the pispark pi example
  spark-submit $SPARK_HOME/examples/src/main/python/pi.py 10

# Finally, start the R front-end (Still experimental and R must be installed) Use "q()" to quit.
 sparkR --master local

Connecting to HDFS
==================
# first start HDFS (if not started)
# based on install from Lesson 2.2, start HDFS and copy a script file to /user/hdfs
# If HDFS is not running, as root run the "hdfs-start.sh" script (see ~/Install-Hadoop-Spark/
Hadoop-Pig-Hive/scripts)
# Start a Spark shell and enter:
  val textFile = sc.textFile("hdfs://localhost:9000/user/hdfs/distribute-exclude.sh")
  textFile.count

# The HDFS URL comes from the ~/hadoop-2.8.1/etc/hadoop/core-site.xml configuration file.
# The line "<value>hdfs://localhost:9000</value>" assigns port 9000 for the HDFS interface.
# don't forget to stop HDFS if you are not using HDFS any further. Use
the "hdfs-stop" script (see Install-Hadoop-Spark/Hadoop-Pig-Hive/scripts/)

Install Zeppelin Web Notebook
=============================
# The following may need some additional configuration for you environment.
  wget -P /tmp http://mirrors.ibiblio.org/apache/zeppelin/zeppelin-0.7.3/zeppelin-0.7.3-bin-
all.tgz

  tar xvzf /tmp/zeppelin-0.7.3-bin-all.tgz -C /opt
```

```
# See this URL
  https://zeppelin.apache.org/docs/0.6.0/install/install.html

# Once Zeppelin is installed, you can change the port for the Zeppelin notebook
  cd /opt/zeppelin-0.7.3-bin-all/conf
  cp zeppelin-site.xml.template zeppelin-site.xml

# Change  port 8080 to 9995
<property>
  <name>zeppelin.server.port</name>
  <value>8080</value>
  <description>Server port.</description>
</property>

# After Change:
<property>
  <name>zeppelin.server.port</name>
  <value>9995</value>
  <description>Server port.</description>
</property>

# Create a Zeppelin user and change ownership (do as root)
  useradd -g hadoop zeppelin
  chown -R zeppelin:hadoop /opt/zeppelin-0.7.3-bin-all

Starting Zeppelin
==================
# use the two scripts to start and stop Zeppelin
  start-zeppelin.sh
  stop-zeppelin.sh

Connect to Zeppelin
====================
# use a web browser to point to "localhost:9995" For example:
  firefox http://localhost:9995

Apache Hadoop and Spark Fundamentals (Third Edition)
LESSON 3.2 USE HDFS COMMAND LINE TOOLS
# To interact with HDFS, the hdfs command must be used.
# HDFS provides a series of commands similar to those found in a standar Unix/Linux
# file system, except they must be prefaced with "hdfs dfs"

List Files in HDFS
------------------
# To list the files in the root HDFS directory enter the following:
$ hdfs dfs -ls /

# To list files in your home directory in HDFS enter the following:
$ hdfs dfs –ls

# The same result can be obtained by issuing a:
$ hdfs dfs -ls /user/student
```

Make a Directory in HDFS

# To make a directory in HDFS use the following command. As with the –ls command, when no path is supplied the users home directory is used. (i.e. /users/student)
$ hdfs dfs -mkdir stuff

Copy Files to HDFS

# To copy a file from your current local directory into HDFS use the following. Note again, that the absence
# of a full path assumes your home directory. In this case the file test is placed in the directory stuff
$ hdfs dfs -put test stuff

# The file transfer can be confirmed by using the –ls command:
$ hdfs dfs -ls stuff

Copy Files from HDFS

# Files can be copied back to your local file system using the following. In this case, the file we copied into HDFS,
# test, will be copied back to the current local directory with the name test-local.
$ hdfs dfs -get stuff/test test-local

Copy Files within HDFS

# The following will copy a file in HDFS.
$ hdfs dfs -cp stuff/test test.hdfs

Delete a File within HDFS

# The following will delete the HDFS file test.dhfs that was created above.
# Note, if the "-skipTrash" argument is not provided, a message indicating
# that the file has been moved to your .Trash directory will be printed.
$ hdfs dfs -rm -skipTrash test.hdfs

Deleted test.hdfs

Delete a Directory in HDFS

# The following will delete the HDFS directory stuff and all its contents.
$ hdfs dfs -rm -r stuff

Apache Hadoop and Spark Fundamentals (Third Edition)
LESSON 3.3 USE HDFS IN PROGRAMS
Notes for Java Based HDFS API
References:
http://wiki.apache.org/hadoop/HadoopDfsReadWriteExample
http://blog.rajeevsharma.in/2009/06/using-hdfs-in-java-0200.html
Tutorial Steps:
mkdir HDFSClient-classes

Compile program
javac -classpath /usr/lib/hadoop/hadoop-core.jar -d HDFSClient-classes HDFSClient.java

```
#Create a java archive file
jar -cvfe HDFSClient.jar org/myorg.HDFSClient -C HDFSClient-classes/ .

# Run the program
hadoop jar ./HDFSClient.jar add ./NOTES.txt /user/student

Apache Hadoop and Spark Fundamentals (Third Edition)
LESSON 3.3 USE HDFS IN PROGRAMS
# Notes for libhdfs HDFS API
# References:
#   http://wiki.apache.org/hadoop/LibHDFS
# Tutorial Steps:
# build the program
export JAVA_LIB=/usr/java/default
gcc hdfs-simple-test.c -I${JAVA_LIB}/include -I${JAVA_LIB}/include/linux -L${JAVA_LIB}/jre/lib/
amd64/server -ljvm -lhdfs -o hdfs-simple-test

# Set Library and Class paths, then run the program
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/java/default/jre/lib/amd64/server/
export CLASSPATH=$(hadoop classpath)

./hdfs-simple-test

Apache Hadoop and Spark Fundamentals (Third Edition)
LESSON 4.1 UNDERSTAND THE MAPREDUCE PARADIGM
# Simplified MapReduce using command line
# get a copy of the file
cp ../../Lesson-3/data/text_file.txt .

# grep is the "mapper" and "wc" is the reducer
grep " Kutuzov " text_file.txt|wc -l

# Now use a simple mapper and reducer script
# Note how data flows in one direction
cat text_file.txt |./mapper.sh |./reducer.sh

Apache Hadoop and Spark Fundamentals (Third Edition)
LESSON 4.1 UNDERSTAND THE MAPREDUCE PARADIGM
# Basic Hadoop java word count example
# Source:
  http://hadoop.apache.org/docs/r0.18.3/mapred_tutorial.html

# Tutorial Steps:
mkdir wordcount_classes

#Compile the WordCount.java program
javac -classpath /usr/lib/hadoop/hadoop-core.jar -d wordcount_classes WordCount.java

#Create a java archive for distribution
jar -cvf wordcount.jar -C wordcount_classes/ .

# Create a directory and move the file into HDFS
hadoop dfs -mkdir /user/student/text_file
```

```
cp ../../Lesson-3/data/text_file.txt .
hadoop dfs -put text_file.txt text_file

# run work count, but first
hadoop jar wordcount.jar  org.myorg.WordCount /user/student/text_file /user/student/text_file-output

# check for output form Hadoop job
hadoop dfs -ls text_file-output

# move it back to working directory (example of "hadoop dfs -get")
hadoop dfs -get text_file-output/part-00000 .

# Note: If you run program again it wont work because /text_file-output exists.
# Hadoop will not overwrite files!
```

Apache Hadoop and Spark Fundamentals (Third Edition)
LESSON 5.1 Use the Streaming Interface

```
# Python Example of using Hadoop Streams interface
# This method will work with any program that can read and write
# to stdin and stdout
# Source:
# http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/

# Start on the command line and map the results.
 echo "foo foo quux labs foo bar quux" | ./mapper.py
# Now sort the results (similar to shuffle)
 echo "foo foo quux labs foo bar quux" | ./mapper.py|sort -k1,1
# Now run the full program with the reduce step
 echo "foo foo quux labs foo bar quux" | ./mapper.py|sort -k1,1|./reducer.py

# If needed, create a directory and move the file into HDFS
hadoop dfs -mkdir /user/student/text_file
cp ../../Lesson-3/data/text_file.txt .
hadoop dfs -put text_file.txt text_file

# make sure output directory is removed from any previous runs
hadoop dfs -rmr text_file-output

# run the following (using the run.sh script may be easier)
hadoop jar /usr/lib/hadoop/contrib/streaming/hadoop-streaming-1.1.2.21.jar \
-file ./mapper.py \
-mapper ./mapper.py \
-file ./reducer.py -reducer ./reducer.py \
-input /user/student/text_file/text_file.txt \
-output /user/student/text_file-output
```

Apache Hadoop and Spark Fundamentals (Third Edition)
LESSON 5.2 Use the Pipes interface

```
# Hadoop C++ Pipes example
# Source:
# http://wiki.apache.org/hadoop/C%2B%2BWordCount
## Compile the program
g++ wordcount.cpp -o wordcount -lhadooppipes -lhadooputils -lpthread -lcrypto
```

```
# If needed, create a directory and move the file into HDFS
hadoop dfs -mkdir /user/student/text_file
cp ../../Lesson-3/data/text_file.txt .
hadoop dfs -put text_file.txt text_file

# put  executable into HDFS so tasktrackers can find the program
hadoop dfs -put wordcount bin
hadoop dfs -rmr text_file-output

# Run program
hadoop pipes \
-D hadoop.pipes.java.recordreader=true  \
-D hadoop.pipes.java.recordwriter=true \
-input text_file   \
-output text_file-output  \
-program bin/wordcount

# Single line version
hadoop pipes -D hadoop.pipes.java.recordreader=true   -D
hadoop.pipes.java.recordwriter=true -input text_file   -output text_file-output  -program bin/
wordcount
```

Apache Hadoop and Spark Fundamentals (Third Edition)
LESSON 5.3 Run the Hadoop grep example

```
# Haddop Grep example
# Source
# http://wiki.apache.org/hadoop/Grep
mkdir Grep_classes

#Compile the WordCount.java program
javac -classpath /usr/lib/hadoop/hadoop-core.jar -d Grep_classes Grep.java

#Create a java archive for distribution
jar -cvf Grep.jar -C Grep_classes/ .

# If needed, create a directory and move the file into HDFS
hadoop dfs -mkdir /user/student/text_file
cp ../../Lesson-3/data/text_file.txt .
hadoop dfs -put text_file.txt text_file

# make sure output dir is gone
hadoop dfs -rmr text_file-output

# Run program
hadoop jar Grep.jar  org.myorg.Grep /user/student/text_file /user/student/text_file-output
Kutuzov
```

Apache Hadoop and Spark Fundamentals (Third Edition)
LESSON 5.6: USING HADOOP VERSION 2 MAPREDUCE FEATURES

```
# Run examples as user (not root or user "hdfs")
# To make the command lines shorter we will create a
# environment define called YARN_EXAMPLES.
```

```
# The YARN_YARN_EXAMPLES define is simply the path to the Hadoop
# examples jar file. This path will will vary by installation
# platform) For example, in the current HDP installation, we
# will use:
  export EXAMPLES=/usr/hdp/2.6.2.0-205/hadoop-mapreduce/

# To find the path on your platform, use the top level
# directory where your Hadoop packages were installed.
  find /opt -name hadoop-mapreduce-examples* -print

  /opt/hadoop-2.8.1/share/doc/hadoop/hadoop-mapreduce-examples
  /opt/hadoop-2.8.1/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.8.1.jar
  /opt/hadoop-2.8.1/share/hadoop/mapreduce/sources/hadoop-mapreduce-examples-2.8.1-
test-sources.jar
  /opt/hadoop-2.8.1/share/hadoop/mapreduce/sources/hadoop-mapreduce-examples-2.8.1-
sources.jar

# Note the path for the hadoop-mapreduce-examples-*.jar file. In this example the Hadoop
version is 2.8.1
# and it is included in all the hadoop jar file names. In this case it is the YARN_EXAMPLES path
is
# /opt/hadoop-2.8.1/share/hadoop/mapreduce/ and the command is::
  export YARN_EXAMPLES=/opt/hadoop-2.8.1/share/hadoop/mapreduce/

# to see all examples (including the famous wordcount example)
yarn jar $YARN_EXAMPLES/hadoop-mapreduce-examples.jar

or

hadoop jar $YARN_EXAMPLES/hadoop-mapreduce-examples.jar

Example 1: Run the pi Example
============================
# Simple MapReduce program to run, good for quick tests.
  yarn jar $YARN_EXAMPLES/hadoop-mapreduce-examples.jar pi 16 100000

Example 2: Run the TestDFSIO Benchmark
=====================================
# Yarn also includes a HDFS benchmark application called TestDFSIO.
# There are three steps.  We will write and read ten 1 GByte files.
# Step 1: Run TestDFSIO in write mode and create data.
yarn jar $YARN_EXAMPLES/hadoop-mapreduce-client-jobclient.jar TestDFSIO -write  -nrFiles
16 -fileSize 1000

# Step 2: Run TestDFSIO in read mode.
yarn jar $YARN_EXAMPLES/hadoop-mapreduce-client-jobclient.jar TestDFSIO -read  -nrFiles
16 -fileSize 1000

# Step 3: Clean up the TestDFSIO data.
yarn jar $YARN_EXAMPLES/hadoop-mapreduce-client-jobclient.jar TestDFSIO -clean

# Interpret the results:
#  There is a results file created in your local directory called "TestDFSIO_results.log"
#  All new results are appended to this file.
```

Example 3: Run the terasort Benchmark
==========
To run the terasort benchmark three separate steps required. In general the
rows are 100 bytes long, thus the total amount of data written is 100 times the
number of rows (i.e. to write a 100 GBytes of data, use 1000000000 rows). You
will also need to specify input and output directories in HDFS.
1. Run teragen to generate 50 GB of data, 500,000,000 rows of random data to sort
yarn jar $YARN_EXAMPLES/hadoop-mapreduce-examples.jar teragen 500000000 /user/hdfs/
TeraGen-50GB

2. Run terasort to sort the database
yarn jar $YARN_EXAMPLES/hadoop-mapreduce-examples.jar terasort /user/hdfs/
TeraGen-50GB /user/hdfs/TeraSort-50GB

3. Run teravalidate to validate the sort Teragen
yarn jar $YARN_EXAMPLES/hadoop-mapreduce-examples.jar teravalidate /user/hdfs/
TeraSort-50GB /user/hdfs/TeraValid-50GB

Interpret the results:
Measure the time it takes to complete the terasort application. Results are
usually reported in Database Size in Seconds (or Minutes).
The perforance can be increased by incresing the number of reducers (default is one)
add the option -Dmapred.reduce.tasks=NUMBER_OF_REDUCERS
The command bleow uses 4 reducers.
yarn jar $YARN_EXAMPLES/hadoop-mapreduce-examples.jar -Dmapred.reduce.tasks=4
terasort /user/hdfs/TeraGen-50GB /user/hdfs/TeraSort-50GB

Don't for get to delete you files in HDFS before the next run!
hdfs dfs -rm -r -skipTrash Tera*

Hadoop Version 2 Log Management
==========
Improved over version 1. To manual enable log aggregation, do the following.
Create the directories in HDFS, as user hdfs
  hdfs dfs -mkdir -p /yarn/logs
  hdfs dfs -chown -R yarn:hadoop /yarn/logs
  hdfs dfs -chmod -R g+rw /yarn/logs

now add the following properties to yarn-site.xml (on all nodes)
and restart yarn (on all nodes)
  <property>
    <name>yarn.nodemanager.remote-app-log-dir</name>
    <value>/yarn/logs</value>
  </property>
  <property>
    <name>yarn.log-aggregation-enable</name>
    <value>true</value>
  </property>

If using Ambari, under the YARN Service Configs as "yarn.log-aggregation-enable"
and is on by default

Command Line Log Management
==========

```
# Logs can be viewed in YARN jobs browser. If command line
# is needed, your can use the "yarn logs" command.
# With log aggregation enabled.  Log management is
# easier. Otherwise, you need to hunt the logs on the nodes
$ yarn logs

# Simple Example with log aggregation on:
# Run the pi example above
# grab all logs and save to file:
$ yarn logs  -applicationId application_1406403419677_0007 > AppOut

# then, to find the container names
grep -B 1 ===== AppOut

# You can zero in on specific container (note:  n0_45454 is  -nodeAddress n0:45454)
yarn logs -applicationId application_1406403419677_0007 -containerId
container_1406403419677_0007_01_000012 -nodeAddress n0:45454|more

Apache Hadoop and Spark Fundamentals (Third Edition)
LESSON 6.1: Demonstrate a Pig example
# Hadoop Pig Examples
# source
#  http://pig.apache.org/docs/r0.10.0/start.html
# Copy the data file to this directory
cp /etc/passwd .

# Copy the data file into HDFS
hadoop dfs -put passwd passwd

# local interactive pig operation
pig -x local

grunt> A = load 'passwd' using PigStorage(':');
grunt> B = foreach A generate $0 as id;
grunt> dump B;

# processing begins
grunt> quit

# local hadoop MapReduce pig operation
pig -x mapreduce

# or just "pig"
grunt> A = load 'passwd' using PigStorage(':');
grunt> B = foreach A generate $0 as id;
grunt> dump B;
# processing begins
grunt> quit

# Pig Script
A = load 'passwd' using PigStorage(':');  -- load the passwd file
B = foreach A generate $0 as id;  -- extract the user IDs
dump B;
```

```
store B into 'id.out'; -- write the results to a file name id.out

# Run the script local
# make sure output directory is gone
/bin/rm -r id.out/
# Run the script
pig -x local id.pig

# Run script through Hadoop mapreduce
# make sure output directory is gone
hadoop dfs -rmr id.out
# Run the script
pig id.pig

Apache Hadoop Fundamentals
LESSON 6.3 APACHE FLUME
Step 1: Download Install Apache Flume
====================================
# assumes HDP repository for yum

  yum install flume flume-agent

# need telent for example
  yum install telnet

Step 2: Simple Test
===================
# agent only needed on nodes that generate data
flume-ng agent --conf conf --conf-file simple-example.conf --name simple_agent -
Dflume.root.logger=INFO,console

# in another window
telnet localhost 44444

Step 3: Web Log Example
=======================
# record the weblogs from the local machine (Ambari output)
# to HDFS using flume. Two file are needed:
  web-server-target-agent.conf - the target flume agent that writes the data to HDFS
  web-server-source-agent.conf - the source flume agent that captures the web log data

# First, as root make local log directory to echo the web log
  mkdir /var/log/flume-hdfs
  chown hdfs:hadoop /var/log/flume-hdfs/

# Next, as user hdfs make a flume data directory in HDFS
  hdfs dfs -mkdir /user/hdfs/flume-channel/

# Start flume target agent as user hdfs (writes data to HDFS)
# This agent should be started before the source agent.
# Note: with HDP flume can be started as a service when the system boots
# e.g "service start flume" The /etc/flume/conf/{flume.conf,flume-env.sh.template}
# files need to be configured. For this example
# /etc/flume/conf/flume.conf would be the same as web-server-target.conf
```

```
 flume-ng agent -c conf -f web-server-target-agent.conf -n collector

# as root, start the source agent to feed the target agent
  flume-ng agent -c conf -f web-server-source-agent.conf -n source_agent

# check to see if working (assuming no errors from flume-ng agents)
# inspect local log, as user hdfs (file name will vary)
  tail -f /var/log/flume-hdfs/1408126765449-1

# inspect data in HDFS (filename will vary)
  hdfs dfs -tail flume-channel/apache_access_combined/140815/FlumeData.1408126868576

Apache Hadoop and Spark Fundamentals (Third Edition)
LESSON 6.2: Demonstrate a Hive example
# source
# http://gettingstarted.hadooponazure.com/hw/hive.html#_Task_3:_Create_1
# http://www.johnandcailin.com/blog/cailin/exploring-apache-log-files-using-hive-and-hadoop
# Make sure metastor is running, start as user hive
nohup hive --service metastore > /var/log/hive/hive.out 2> /var/log/hive/hive.log &

# start hive
hive

# simple test
hive> CREATE TABLE pokes (foo INT, bar STRING);
hive> SHOW TABLES;
hive> DROP TABLE pokes;

# create, load and run a query
hive> CREATE TABLE logs(t1 string, t2 string, t3 string, t4 string, t5 string, t6 string, t7 string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ' ';
hive> LOAD DATA LOCAL INPATH 'sample.log' OVERWRITE INTO TABLE logs;
hive> SELECT t4 AS sev, COUNT(*) AS cnt FROM logs WHERE t4 LIKE '[%' GROUP BY t4;

# exit hive
hive> exit;

Apache Hadoop and Spark Fundamentals (Third Edition)
LESSON 6.3 APACHE OOZIE
Step 1: Set hadoop.proxyuser.oozie.groups in core-site.xml
=========================================================
# This step depends on the group that will be running
# Oozie. The default is "users." We are going to run
# Oozie as group "hadoop" so there is a change needed
# in the core-site.xml file.
# If using Ambari, go to Services/HDFS/Config tab and at the bottom
# click on  Custom core-site.xml. Change "hadoop.proxyuser.oozie.groups"
# from "users" to "hadoop."
# Note: You may wish to change this for faclon, hcat, and hive as well.
# If you plan on keeping placing users in group "users" this is not needed.
# If not using Ambari, set this directly in core-site.xml and restart Hadoop.
<property>
     <name>hadoop.proxyuser.oozie.groups</name>
     <value>hadoop</value>
```

</property>

Step 2: Download Oozie Examples
==========
The examples are part of the oozie-client-4.0.0.2.1.2.1-471.el6.noarch.rpm
package. Make sure this is installed, then as user hdfs,
  tar xvzf /usr/share/doc/oozie-4.0.0.2.1.2.1/oozie-examples.tar.gz

the examples must also be placed in HDFS
  hdfs dfs -put examples/ examples

The the Oozie Share Library must be installed in HDFS.
If you are using the Ambari install of HDP 2.1 from lesson 8.6, this is
already in HDFS under: /user/oozie/share/lib. It is part of the
oozie-4.0.0.2.1.2.1-471.el6.noarch.rpm and is installed on the host in
/usr/lib/oozie/oozie-sharelib.tar.gz. If you install by hand, then
make /user/oozie in HDFS and put the extracted oozie-sharelib files
in this directory as user oozie and group hadoop.
The example applications are under the examples/app directory, one directory per example.
The directory contains the application XML file (workflow, or worklfow and coordinator),
the job.properties file to submit the job and any JAR files the example may need.
The inputs for all examples are in the examples/input-data/ directory.
The examples create output under the examples/output-data/${EXAMPLE_NAME} directory.

Step 3: Run The Simple MapReduce Example
==========
job.properties - defined values (path names, ports, etc) for a jobs
workflow.xml - workflow for the job. In this case, simple MR (pass/fail)
view the workflow.xml
  vi examples/apps/map-reduce/workflow.xml

update local job.properties to reflect current system.
Important: This must be done for all the examples.
   vi examples/apps/map-reduce/job.properties

# change
   nameNode=hdfs://localhost:8020
   jobTracker=localhost:8032

# to
   nameNode=hdfs://limulus:8020
   jobTracker=limulus:8050

Note the change in port number for the jobTracker line.
Run the job
oozie job -run -oozie http://limulus:11000/oozie -config examples/apps/map-reduce/
job.properties

To avoid having to provide the -oozie option with the Oozie URL with every oozie command,
set OOZIE_URL env variable to the Oozie URL in the shell environment. For example:
   export OOZIE_URL="http://limulus:11000/oozie"

Then the command is (it then prints the job id number):

```
  oozie job  -config examples/apps/map-reduce/job.properties -run
  job: 0000005-140824145757925-oozie-oozi-W

  oozie job  -info job: 0000005-140824145757925-oozie-oozi-W

# Use the Oozie web console to view job progress and flow.
  http://limulus:11000/oozie/

Step 4: Run the Oozie Demo Application
=====================================
# A more sophisticated example can be found in the demo directory:
# View the workflow.xml
  vi examples/apps/demo/workflow.xml

# Don't forget to change examples/apps/demo/job.properties (see above)
# To run
 oozie job -run   -config examples/apps/demo/job.properties

# Short Summary of Oozie job Commands
====================================
1) Run Job:
  oozie job -run  -config JOB_PROPERITES

2) Submit job:
  oozie job -config JOB_PROPERTIES -submit job: OOZIE_JOB_ID

3) Run job:
  oozie job  -start OOZIE_JOB_ID

4) Check the status:
  oozie job  -info OOZIE_JOB_ID

5) Suspend workflow:
  oozie job  -suspend OOZIE_JOB_ID

6) Resume workflow:
  oozie job  -resume OOZIE_JOB_ID

7) Re-run workflow:
  oozie job  -config JOB_PROPERTIES -rerun OOZIE_JOB_ID

8) Should you need to kill the job:
  oozie job  -kill OOZIE_JOB_ID

9) View server logs:
  oozie job  -logs OOZIE_JOB_ID

# Full Logs are available at:
  /var/log/oozie on the Oozie server.

Apache Hadoop Fundamentals
LESSON 6.4 APACHE SQOOP
Step 1: Download and Load Sample MySQL Data
==========================================
```

```
# Assume mysql is installed and working on the host
# Ref: http://dev.mysql.com/doc/world-setup/en/index.html
# Get the database:
  wget http://downloads.mysql.com/docs/world_innodb.sql.gz

# Load into MySQL
   mysql -u root -p
   mysql> CREATE DATABASE world;
   mysql> USE world;
   mysql> SOURCE world_innodb.sql;
   mysql> SHOW TABLES;

# To see table details:
   mysql> SHOW CREATE TABLE Country;
   mysql> SHOW CREATE TABLE City;
   mysql> SHOW CREATE TABLE CountryLanguage;


Step 2: Add Sqoop User Permissions for Local Machine and Cluster
================================================================
   mysql> GRANT ALL PRIVILEGES ON world.* To 'sqoop'@'limulus' IDENTIFIED BY 'sqoop';
   mysql> GRANT ALL PRIVILEGES ON world.* To 'sqoop'@'10.0.0.%' IDENTIFIED BY 'sqoop';
   mysql> quit

# Login as sqoop to test
   mysql -u sqoop -p
   mysql> USE world;
   mysql> SHOW TABLES;

   mysql> quit

Step 3: Import Data Using Sqoop
===============================
# Use Sqoop to List Databases
   sqoop list-databases --connect jdbc:mysql://limulus/world --username sqoop --password
sqoop

# List Tables
   sqoop list-tables --connect jdbc:mysql://limulus/world --username sqoop --password sqoop

# Make directory for data
   hdfs dfs -mkdir sqoop-mysql-import

# Do the import
# -m is number of map tasks
  sqoop import --connect jdbc:mysql://limulus/world  --username sqoop --password sqoop --
table Country  -m 1 --target-dir /user/hdfs/sqoop-mysql-import/country

   hdfs dfs -ls sqoop-mysql-import/country

   hdfs dfs  -cat sqoop-mysql-import/country/part-m-00000

# Using and Options File
#   Can use and options file to avoid rewriting same options
```

```
#   Example (vi world-options.txt):
  import
  --connect
  jdbc:mysql://limulus/world
  --username
  sqoop
  --password
  sqoop

sqoop  --options-file world-options.txt --table City  -m 1 --target-dir /user/hdfs/sqoop-mysql-
import/city

# First use a single mapper "-m 1" The $CONDITIONS is requried by WHERE, leave blank.
  sqoop  --options-file world-options.txt -m 1 --target-dir /user/hdfs/sqoop-mysql-import/
canada-city --query "SELECT ID,Name from City WHERE CountryCode='CAN' AND \
$CONDITIONS"

  hdfs dfs  -cat sqoop-mysql-import/canada-city/part-m-00000

# Using Multiple Mappers
# The $CONDITIONS variable is needed for more than one mapper.
# If you want to import the results of a query in parallel, then each map task will need
# to execute a copy of the query, with results partitioned by bounding conditions inferred
# by Sqoop. Your query must include the token $CONDITIONS which each Sqoop process will
# replace with a unique condition expression based on the "--split-by" option.
# You may need to select another splitting column with --split-by option if your
# primary key is not uniformly distributed.
# Since -m 1 is one map, we don't need to specify a --split-by option.
# Now use multiple mappers, clear resutls from previous import.
  hdfs dfs -rm -r -skipTrash  sqoop-mysql-import/canada-city

  sqoop  --options-file world-options.txt -m 4 --target-dir /user/hdfs/sqoop-mysql-import/
canada-city --query "SELECT ID,Name from City WHERE CountryCode='CAN' AND \
$CONDITIONS" --split-by ID

  hdfs dfs -ls  sqoop-mysql-import/canada-city

Step 4: Export Data from HDFS to MySQL
======================================
# Create table for exported data, also need staging table

  mysql> CREATE TABLE `CityExport` (
        `ID` int(11) NOT NULL AUTO_INCREMENT,
        `Name` char(35) NOT NULL DEFAULT '',
        `CountryCode` char(3) NOT NULL DEFAULT '',
        `District` char(20) NOT NULL DEFAULT '',
        `Population` int(11) NOT NULL DEFAULT '0',
        PRIMARY KEY (`ID`));

  mysql> CREATE TABLE `CityExportStaging` (
        `ID` int(11) NOT NULL AUTO_INCREMENT,
        `Name` char(35) NOT NULL DEFAULT '',
        `CountryCode` char(3) NOT NULL DEFAULT '',
        `District` char(20) NOT NULL DEFAULT '',
```

```
        `Population` int(11) NOT NULL DEFAULT '0',
        PRIMARY KEY (`ID`));

  sqoop --options-file cities-export-options.txt --table CityExport  --staging-table
CityExportStaging  --clear-staging-table -m 4 --export-dir /user/hdfs/sqoop-mysql-import/city

# Check table in Mysql
  mysql> select * from CityExport limit 5;

Some Handy Clean-up Commands
============================
# remove the table
  mysql> Drop table `CityExportStaging`;

# remove data in table
  mysql> delete from CityExportStaging;
  mysql> delete from CityExportStaging;

# clean-up imported files
hdfs dfs -rm -r  -skipTrash sqoop-mysql-import/{country,city, canada-city}

Apache Hadoop and Spark Fundamentals (Third Edition)
Lesson 7.1 LEARN SPARK LANGUAGE BASICS
Starting an Interactive Spark Shell
==================================
# Start with Scala front end (default) (:q or :quit to exit)
# https://spark.apache.org/docs/1.6.2/quick-start.html
spark-shell

# Start with R front end (q() to exit)
# https://spark.apache.org/docs/1.6.2/sparkr.html
sparkR

# Start with Python front end (quit() or ctrl-D to exit)
# https://spark.apache.org/docs/1.6.2/quick-start.html
pyspark

Interactive Example with PySpark
================================
# First quiet down the logging:
# Valid log levels include: ALL, DEBUG, ERROR, FATAL, INFO, OFF, TRACE, WARN
sc.setLogLevel("WARN")

# RDDs have actions, which return values, and transformations, which return pointers
# to new RDDs. Spark performs its calculations, it will not execute any of
# the transformations until an action occurs.
# Read a text file from local file systems (TRANSFORMATION)
text=sc.textFile("file:///home/deadline/Hadoop_Fundamentals_Code_Notes-V3/Lesson-7/
Lesson-7.1/text_file.txt")

# Read a text file from HDFS (TRANSFORMATION)
text=sc.textFile("Text_file/text_file.txt")

# Count number f lines (ACTION)
```

```
text.count()

Mapping and Reducing
====================
# Word count in spark looks like the following:
# counts = text.flatMap(lambda line: line.split(" ")). \
#          map(lambda word: (word, 1)). \
#          reduceByKey(lambda a, b: a + b)
# Consider a simple test file  with a couple of lines of text:
cat map-text.txt

# read in file:
MapText=sc.textFile("file:///home/deadline/Hadoop_Fundamentals_Code_Notes-V3/Lesson-7/
Lesson-7.1/map-text.txt")

# inspect file as read:
MapText.collect()

# now do map to split on spaces, result is lines broken into words (elements) and
# sentences still exist (lists)
mp=MapText.map(lambda line:line.split(" "))
mp.collect()

# now do flatMap to split on spaces, result is lines broken on words (elements) and
# sentences are gone (flattened to one big list)
fm=MapText.flatMap(lambda line:line.split(" "))
fm.collect()

# apply operation to each map
# flatMap(lambda line: line.split(" "))

Reducing
========
# Combine values with the same key (a is the the accumulator)
# reduceByKey(lambda a, b: a + b)

Word Count
==========
# Put it all together and count all words, split on spaces, i.e. spaces define a word
counts = text.flatMap(lambda line: line.split(" ")). \
        map(lambda word: (word, 1)). \
        reduceByKey(lambda a, b: a + b)

# look at first 5 elements
counts.take(5)

# Save results locally (also in HDFS if needed)
counts.saveAsTextFile("file:///home/deadline/Hadoop_Fundamentals_Code_Notes-V3/
Lesson-7/Lesson-7.1/counts-result")

# Modify for single word search "Kutuzov"
counts = text.flatMap(lambda line: line.split(" ")). \
        filter(lambda w: "Kutuzov" in w). \
        map(lambda word: (word, 1)). \
```

```
        reduceByKey(lambda a, b: a + b)

# Show all results
counts.collect()

# print results using for loop
for x in counts.collect(): print x

# add a map to replace "," note how (u'Kutuzov,', 68) is now gone
# and (u'Kutuzov', 395) has increased.

counts = text.flatMap(lambda line: line.split(" ")). \
        filter(lambda w: "Kutuzov" in w). \
        map(lambda x: x.replace(',','')). \
        map(lambda word: (word, 1)). \
        reduceByKey(lambda a, b: a + b)

for x in counts.collect(): print x

Apache Hadoop and Spark Fundamentals (Third Edition)
Lesson 7.2 DEMONSTRATE A PYSPARK COMMAND LINE EXAMPLE
A Stand Alone Pi Estimator
==========================
# Needs numpy installed for random function.
# Includes an example of how to uses a Python function
# Also note the use of "sc = SparkContext(appName = "test")"
# to set spark context.
# Uses parallelize() function to break up count function across available executors
cat pi-estimate.py

from pyspark import SparkContext
from numpy import random
sc = SparkContext(appName = "test")
sc.setLogLevel("WARN")

n=5000000
def sample(p):
   x, y = random.random(), random.random()
   return 1 if x*x + y*y < 1 else 0

count = sc.parallelize(xrange(0,n)). \
          map(sample). \
          reduce(lambda a, b: a + b)

print "Pi is roughly %f" % (4.0 * count / n)

# Pick random points in the unit square ((0, 0) to (1,1)) and see how many
# fall in the unit circle. The fraction should be pi/4
# Run as stand alone:
spark-submit pi-estimate.py

# Lesson 8.1 IMPORT DATA INTO HIVE TABLES
# Example commands for loading CSV data into a HIVE table
```

```
# Assumes path is for user "deadline," change all references from
# /user/deadline to your path in HDFS
# create the names directly in HDFS
 hdfs dfs -mkdir names

# move names to HDFS
 hdfs dfs -put names.csv names

# start HIVE
  hive

# HIVE commands to create external HIVE table
CREATE EXTERNAL TABLE IF NOT EXISTS Names_text(
EmployeeID INT, FirstName STRING, Title STRING, State STRING, Laptop STRING)
COMMENT 'Employee Names'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
location '/user/deadline/names';

# check to see if names are there:
Select * from Names_text limit 5;

# now create HIVE table
CREATE TABLE IF NOT EXISTS Names(
EmployeeID INT, FirstName STRING, Title STRING, State STRING, Laptop STRING)
COMMENT 'Employee Names'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS ORC;

# copy data from external table to internal table
INSERT OVERWRITE TABLE Names SELECT * FROM Names_text;

# check to see if names are there:
Select * from Names limit 5;

# create partitioned table
CREATE TABLE IF NOT EXISTS Names_part(
EmployeeID INT, FirstName STRING, Title STRING, Laptop STRING)
COMMENT 'Employee names partitioned by state'
PARTITIONED BY (State STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS ORC;

# Turn off dynamic partition strict mode (requires at least one static partition column)
hive> SET hive.exec.dynamic.partition.mode = nonstrict;

# create partitioned (by State) table
INSERT INTO TABLE Names_part PARTITION(state)
SELECT EmployeeID, FirstName, Title, Laptop, State FROM Names_text;
```

```
# check to see if names are there:
Select * FROM Names_part WHERE State='DE';

# Create table using Parquet format
CREATE TABLE IF NOT EXISTS Names_Parquet(
EmployeeID INT, FirstName STRING, Title STRING, State STRING, Laptop STRING)
COMMENT 'Employee Names'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS PARQUET;

# add data
INSERT OVERWRITE TABLE Names_Parquet SELECT * FROM Names_text;

# save external table in Parquet format
CREATE EXTERNAL TABLE IF NOT EXISTS Names_Parquet_Ext(
EmployeeID INT, FirstName STRING, Title STRING, State STRING, Laptop STRING)
COMMENT 'Employee Names'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS PARQUET
LOCATION '/user/deadline/names-parquet';

# add data
INSERT OVERWRITE TABLE Names_Parquet_Ext SELECT * FROM Names_text;

# Lesson 8.2 USE SPARK TO IMPORT DATA INTO HDFS
# Example commands to enter data (CSV/JSON) into Spark using PySpark
# Start PiSpark
  pyspark

# Use quit() or Ctrl-D to exit
# Import some functions:
from pyspark.sql import SQLContext
from pyspark.sql.types import *
from pyspark.sql import Row

# Create the SQL context
sqlContext = SQLContext(sc)

#read in CSV data
csv_data = sc.textFile("file:////home/deadline/Hadoop_Fundamentals_Code_Notes-V3/
Lesson-8/Lesson-8.2/names.csv")

# confirm RDD
type(csv_data)

<class 'pyspark.rdd.RDD'>

# peak at the RDD data
csv_data.take(5)

# Split on comma, and see difference
```

```
csv_data  = csv_data.map(lambda p: p.split(","))
csv_data.take(5)

# remove the header
header=csv_data.first()
csv_data = csv_data.filter(lambda p:p != header)

# Place RDD into Spark DataFrame
df_csv = csv_data.map(lambda p: Row(EmployeeID = int(p[0]), FirstName = p[1], Title=p[2],
State=p[3], Laptop=p[4])).toDF()

# show the DataFrame format
df_csv

# show the first 5 rows of the DataFrame
df_csv.show(5)

#print the DataFrame schema
df_csv.printSchema()

# Create and Write a Hive External and Internal Table in pySpark
# Similar to using Hive, just call Hive commands
from pyspark.sql import HiveContext
sqlContext = HiveContext(sc)

sqlContext.sql("CREATE TABLE IF NOT EXISTS MoreNames_text(EmployeeID INT, FirstName
STRING, Title STRING, State STRING, Preference STRING) ROW FORMAT DELIMITED FIELDS
TERMINATED BY ',' LINES TERMINATED BY '\n' ")

sqlContext.sql("CREATE TABLE IF NOT EXISTS MoreNames(EmployeeID INT, FirstName
STRING, Title STRING, State STRING, Preference STRING) ROW FORMAT DELIMITED FIELDS
TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS ORC")

sqlContext.sql("LOAD DATA LOCAL INPATH 'MoreNames.txt' INTO TABLE MoreNames_text")

sqlContext.sql("INSERT OVERWRITE TABLE MoreNames SELECT * FROM MoreNames_text")

result = sqlContext.sql("FROM MoreNames SELECT EmployeeID, FirstName, State")

result.show(5)

# Read a Hive table from Spark
# Only read EmplyeeID and Laptop

sqlContext = HiveContext(sc)
df_hive = sqlContext.sql("SELECT EmployeeID, Laptop  FROM names")

df_hive.show(5)

df_hive.printSchema()

# Read from JSON
# Please note Spark expects each line to be a separate JSON object,
# so it will fail if you'll try to load a pretty formatted JSON file.
```

```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
df_json = sqlContext.read.json("file:////home/deadline/Hadoop_Fundamentals_Code_Notes-
V3/Lesson-8/Lesson-8.2/names.json")
df_json.show()

df_json

df_json.printSchema()

Apache Hadoop and Spark Fundamentals (Third Edition)
Lesson 8.2 USE SPARK TO IMPORT DATA INTO HDFS
# Example commands to enter data (CSV/JSON) into Spark using PySpark
# Start PiSpark
  pyspark

# Use quit() or Ctrl-D to exit
# Import some functions:
from pyspark.sql import SQLContext
from pyspark.sql.types import *
from pyspark.sql import Row

# Create the SQL context
sqlContext = SQLContext(sc)

#read in CSV data
csv_data = sc.textFile("file:////home/deadline/Hadoop_Fundamentals_Code_Notes-V3/
Lesson-8/Lesson-8.2/names.csv")

# confirm RDD
type(csv_data)

# peak at the RDD data
csv_data.take(5)

# Split on comma, and see difference
csv_data  = csv_data.map(lambda p: p.split(","))
csv_data.take(5)

# remove the header
header=csv_data.first()
csv_data = csv_data.filter(lambda p:p != header)

# Place RRD into Spark DataFrame
df_csv = csv_data.map(lambda p: Row(EmployeeID = int(p[0]), FirstName = p[1], Title=p[2],
State=p[3], Laptop=p[4])).toDF()

# show the DataFrame format
df_csv

# show the first 5 rows of the DataFrame
df_csv.show(5)
```

```
#print the DataFrame schema
df_csv.printSchema()

# Create and Write a Hive External and Internal Table in pySpark
# Similar to using Hive, just call Hive commands
from pyspark.sql import HiveContext
sqlContext = HiveContext(sc)

sqlContext.sql("CREATE TABLE IF NOT EXISTS MoreNames_text(EmployeeID INT, FirstName
STRING, Title STRING, State STRING, Preference STRING) ROW FORMAT DELIMITED FIELDS
TERMINATED BY ',' LINES TERMINATED BY '\n' ")

sqlContext.sql("CREATE TABLE IF NOT EXISTS MoreNames(EmployeeID INT, FirstName
STRING, Title STRING, State STRING, Preference STRING) ROW FORMAT DELIMITED FIELDS
TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS ORC")

sqlContext.sql("LOAD DATA LOCAL INPATH 'MoreNames.txt' INTO TABLE MoreNames_text")

sqlContext.sql("INSERT OVERWRITE TABLE MoreNames SELECT * FROM MoreNames_text")

result = sqlContext.sql("FROM MoreNames SELECT EmployeeID, FirstName, State")

result.show(5)

# Read a Hive table from Spark
# Only read EmplyeeID and Laptop

sqlContext = HiveContext(sc)
df_hive = sqlContext.sql("SELECT EmployeeID, Laptop  FROM names")

df_hive.show(5)

df_hive.printSchema()

# Read from JSON
# Please note Spark expects each line to be a separate JSON object,
# so it will fail if you'll try to load a pretty
# formatted JSON file.

from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
df_json = sqlContext.read.json("file:////home/deadline/Hadoop_Fundamentals_Code_Notes-
V3/Lesson-8/Lesson-8.2/names.json")
df_json.show()

df_json

df_json.printSchema()

Apache Hadoop and Spark Fundamentals (Third Edition)
LESSON 10.1: APACHE AMBARI 2.5 INSTALL OF APACHE HADOOP (part 1)
# We will be using Ambari to install Hadoop on a small 4-node cluster.
# The cluster will have one "login" node named "limulus"
# and three worker nodes named "n0, n1, n2"
```

```
# There are two steps:
#  1. Configure nodes and start ambari-agents and start ambari-server (text mode)
#  2. Switch to web browser and use Ambari

Step 1: Install Java (if needed) and PDSH
=========================================
# we are using OpenJDK 1.7, usually installed with Red Hat/CentOS
# If using RHEL (or variant) use java-1.7.0-openjdk RPM (install if needed,
# as root, "yum install java-1.7.0-openjdk-devel java-1.7.0-openjdk"
# Add java.sh to profile.d
 echo 'export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk.x86_64'> /etc/profile.d/java.sh

 source /etc/profile.d/java.sh

# Java needs to be present on all cluster nodes, this can be accomplished through a
# kickstart file.

# To make installation easier, the parallel distributed shell (pdsh) package
# is installed. We also add the "epel" repository so we can find the pdsh package.
# On admin, main or headnode install epel repo and pdsh
# In general there will be a node from which you will run the main
# Ambari server. We will refer to this as the headnode.
 rpm -Uvh  http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm

 yum install pdsh-rcmd-ssh

# To have pdsh work effectively, each host in the cluster must support password-less
# root login. This task is accomplished by placing the head node public ssh key
# in the .ssh directory in each of the worker nodes ~/.ssh/authorized_keys file
# If provisioning nodes with a "kickstart" install, this is easily done by
# writing the key in the post install section.


Step 2: Set up the Ambari Server and Agents
===========================================
# Use wget to install the Ambari repo on the head node
# Check on https://docs.hortonworks.com for latest version
 wget -nv http://public-repo-1.hortonworks.com/ambari/centos6/2.x/updates/2.5.1.0/
ambari.repo -O /etc/yum.repos.d/ambari.repo

# Use pdsh to install Ambari repo on the worker nodes (run "man pdsh" for options
explanation):
 pdsh -w n[0-2] wget -nv http://public-repo-1.hortonworks.com/ambari/centos6/2.x/updates/
2.5.1.0/ambari.repo -O /etc/yum.repos.d/ambari.repo

# install the Ambari Agent
 pdsh -w n[0-2] yum -y install ambari-agent

# Set the Ambari Server hostname on nodes (limulus in this case)
  pdsh -w n[0-2] "sed -i 's/hostname=localhost/hostname=limulus/g' /etc/ambari-agent/conf/
ambari-agent.ini"

# Start the Ambari Agents
  pdsh -w n[0-2] "service ambari-agent start"
```

```
# We double up the head node as a worker node, so install ambari agent and server
  yum install ambari-agent ambari-server

# set the Ambari host to itself
 sed -i 's/hostname=localhost/hostname=limulus/g' /etc/ambari-agent/conf/ambari-agent.ini

# Start the local ambari agent
  service ambari-agent start

# Set up the ambari server to use local openjdk
  ambari-server setup -j /usr/lib/jvm/java-1.7.0-openjdk.x86_64

# Start the ambari server
  ambari-server start

Step 3: Go to web browser and login
==================================
  firefox http://headnode:8080/

# login="admin" password="admin"

Step 4: If needed, Clean-up
===========================
# To clean out the install use "reset"
# !!! It will wipe out database and require reinstall
  ambari-server stop
  ambari-server reset

# To remove RPMS from nodes:
# !!! Careful: May delete accounts as well as RPMS
# Often not needed if doing a reset "reinstall"

 python /usr/lib/python2.6/site-packages/ambari_agent/HostCleanup.py

EXTRA STEP IF USING FALCON
==========================
# Falcon fix: (Hortonworks cannot ship this jar, so it must be downloaded separately)
# Use thiese steps
  wget http://search.maven.org/remotecontent?filepath=com/sleepycat/je/5.0.73/je-5.0.73.jar -
O /usr/hdp/current/falcon-server/server/webapp/falcon/WEB-INF/lib/je-5.0.73.jar
  chown falcon:hadoop /usr/hdp/current/falcon-server/server/webapp/falcon/WEB-INF/lib/
je-5.0.73.jar

Apache Hadoop Fundamentals
LESSON 10.3: PERFORM SIMPLE ADMINISTRATION AND MONITORING USING THE
COMMAND LINE
# We will be using a small cluster to demonstrate some basic command line functions
# The following list of tasks is only a quick summary of some common tasks.
# There are many other aspects to Hadoop and Spark administration (most of
# which can be done with Ambari)
```

Managing YARN Jobs
==========
YARN jobs can be managed using the yarn application command.
yarn application -help

usage: application
 -appStates <States>   Works with -list to filter applications
                                based on input comma-separated list of
                                application states. The valid application
                                state can be one of the following:
                                ALL,NEW,NEW_SAVING,SUBMITTED,ACCEPTED,RUN
                                NING,FINISHED,FAILED,KILLED
 -appTypes <Types>   Works with -list to filter applications
                                based on input comma-separated list of
                                application types.
 -help   Displays help for all commands.
 -kill <Application ID>   Kills the application.
 -list   List applications. Supports optional use
                                of -appTypes to filter applications based
                                on application type, and -appStates to
                                filter applications based on application
                                state.
 -movetoqueue <Application ID> Moves the application to a different
                                queue.
 -queue <Queue Name>   Works with the movetoqueue command to
                                specify which queue to move an
                                application to.
 -status <Application ID>   Prints the status of the application.

The following shows a listing for a MAPREDUCE job running on the cluster.
MapReduce jobs can also be controlled with the mapred job command.
  yarn application -list

To kill and application use the Application-Id
  yarn application -kill application_1508525544406_0003

To see all finished applications
  yarn application -list -appStates FINISHED

To see all finished SPARK jobs
  yarn application -list -appTypes SPARK -appStates FINISHED

Adding Users to HDFS
==========
Must be done as user hdfs
Keep in mind errors running Hadoop applications are often due to file permissions.
To quickly create user accounts manually on a Linux based system, perform the following:
1. Add the user to the group for your OS on the HDFS client system. In most cases,
groupname should be the hdfs superuser user, which is often "hadoop" or "hdfs."
        useradd -G <groupname> <username>

2. Create the username directory in HDFS.
        hdfs dfs -mkdir /user/<username>

```
#   3. Give that account ownership over its directory in HDFS.
      hdfs dfs -chown <username>:<groupname> /user/<username>

Perform an FSCK on HDFS
=======================
# Must be done as user hdfs
# The health of HDFS is checked by using the hdfs fsck <path> (file system check) command.
# The entire HDFS name space can be checked, or a subdirectory can be entered as an
argument to the command.
# The following example checks the entire HDFS namespace.
  hdfs fsck /

# To remove corrupted files
  hdfs fsck -delete /user

# To move corrupted files to /lost+found (in HDFS)
  hdfs fsck -move /

Generate HDFS Report
====================
# Must be done as user hdfs -- all others get an abbreviated report
# Similar information on HDFS web GUI
  hdfs dfsadmin -report

# Get a quick look at HDFS "spread"
  hdfs dfsadmin -report|grep "DFS Used%"

Balance HDFS
============
# Must be done as user hdfs
# HDFS may become un balanced. Run the following to get all data nodes storage levels within
10% of each other
# For lower or higher percentage use 1=1% 2=2% 5=5% etc.
  hdfs balancer -threshold 10

# You can set upper limit on bandwidth. Balancing can be stopped and restarted at any time.

HDFS Safe Mode
==============
# Must be done as user hdfs
# When the NameNode starts, it loads the file system state from the fsimage and then applies
the edits log file.
# It then waits for DataNodes to report their blocks. During this time, the NameNode stays in a
read-only safemode.
# The NameNode leaves safemode automatically after the DataNodes have reported that most
file system blocks are available.
# No changes are allowed when HDFS is in safe mode. The administrator can place HDFS in
safemode using the
# following command:
    hdfs dfsadmin -safemode enter

# Entering the following can turn off safemode:
   hdfs dfsadmin -safemode leave
```

HDFS may drop into safe mode if there is a major issue with the file system (e.g. a full DataNode).
The file system will not leave safemode until the situation is resolved.
To check whether HDFS is in safemode, enter the following command.
  hdfs dfsadmin -safemode get

LESSON 10.4: UTILIZE ADDITIONAL FEATURES OF HDFS
We will be using a small cluster to demonstrate some basic HDFS features
Snapshots
Web GUI
NFSv3 gateway to HDFS

HDFS Snapshots
==========
HDFS Snapshots are read-only, point-in-time copies of HDFS. Snapshots
can be taken on a sub tree of the file system or the entire file system.
Some common use cases of snapshots are data backup, protection against
user errors and disaster recovery.
Snapshots can be taken on any directory once the directory has been set as
"snapshot-able." (Up to 65,536 simultaneous snapshots.)
Administrators may set any directory to be snapshottable.
Nested snapshottable directories are currently not allowed.
 hdfs dfsadmin -allowSnapshot /user/deadline/Text_file

Once the directory has been made snapshottable, the snapshot can be taken with
the following command.
The command requires the directory path and a name for the snapshot:
 hdfs dfs -createSnapshot /user/deadline/Text_file WaP-snap-1

Check the contents of the directory
  hdfs dfs -ls /user/deadline/Text_file

If the file is deleted, it can be restored from the snapshot:
 hdfs dfs -rm -skipTrash /user/deadline/Text_file/text_file.txt

Deleted /user/hdfs/text_file-input/text_file.txt
  hdfs dfs -ls /user/deadline/Text_file

The restoration process is basically a simple copy from the .snapshot directory
to the previous directory (or anywhere else).
  hdfs dfs -cp /user/deadline/Text_file/.snapshot/WaP-snap-1/text_file.txt /user/deadline/Text_file/

A snapshot can be deleted using the following command:
 hdfs dfs -deleteSnapshot /user/deadline/Text_file WaP-snap-1

Finally, a directory can be made un-snapshot-able by using the
following command:
 hdfs dfsadmin -disallowSnapshot /user/deadline/Text_file

Configuring an NFSv3 Gateway to HDFS
==========
HDFS supports an NFS version 3 (NFSv3) gateway. This feature enables files

```
# to be easily moved between HDFS and client systems. The NFS Gateway supports
# NFSv3 and allows HDFS to be mounted as part of the client's local file system.
# In Lesson 10.2 we added HDFS NFS gateway to node "n0"
# The following commands illustrate the gateway usage
  mkdir /mnt/hdfs

  mount -t nfs -o vers=3,proto=tcp,nolock n0:/  /mnt/hdfs/

  ls /mnt/hdfs

  hdfs dfs -ls /

# test the gateway
  cp /mnt/hdfs/user/deadline/Text_file/text_file.txt copy-of-text_file.txt

  cp copy-of-text_file.txt /mnt/hdfs/user/deadline/Text_file/

  ls -l /mnt/hdfs/user/deadline/

# node n0 uses /tmp/.hdfs-nfs for staging

NOTE:
=====
# If when copying files to and from the NFSv3 gateway, you get
# "cp: cannot create regular file `....': Input/output error"
# you may need to set HDFS Access time precision (dfs.namenode.accesstime.precision)
# to 3600000 (default is 0) in the HDFS General settings.
```