Hadoop for Java Developers [HOL1813]

by Christopher M. Judd (javajudd@gmail.com)

Contents

Hands-On Labs

This hands-on lab assumes you are logged into a virtual machine that completed the Appendix A - Environment Setup as the user hduser/hduser.

Lab 1 - Run First Hadoop Job

This lab runs the pi example provided with Hadoop. This example calculates pi by using the Quasi-Monte Carlos method which does not give you an exact value of pi but instead uses a sample set of randomly generated numbers and an equation to approximate pi. Therefore when

you run this job, you will specify how many sample sets to run and how many random numbers you want the job to generate in each sample set.

The goals of this lab are to:

- insure Hadoop is install properly
- learn how to run a Hadoop job in local standalone mode
- discover what examples are included in Hadoop

1. Run the pi example with 4 sets of data containing 1000 random samples in each.
   `$ hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.2.0.jar pi 4 1000`
2. What is the estimated value of pi?
3. To see other examples included in Hadoop run the same examples jar with no parameters.
   `$ hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.2.0.jar`
4. What other examples are there?
5. To see the parameter options an example run the example jar with the name of example. `$ hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.2.0.jar pi`

Lab 2 - Run Hadoop in a Pseudo-Distributed mode

Hadoop's strength is it can run in a distributed manor across on commondity hardware. This lab will simulate running Hadoop in a distributed manor.

The goals of this lab are to:

- be able to start and stop YARN
- run a Hadoop job in a distributed manor
- view node activity

1. Edit a MapReduce config file and remove commented out XML.
   `$ sudo vim $HADOOP_HOME/etc/hadoop/mapred-site.xml`

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

2. Start YARN.
   `$ start-yarn.sh`
3. Verify YARN is running and that you see the NodeManager and ResourceManager processes.
   `$ jps`
4. Run the same pi example as earilier.
   `$ hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.2.0.jar pi 4 1000`
5. View the applications/jobs and their containers while the job is running. Open a browser and navigate to http://localhost:8042/node.
6. Click on List of Applicaitons in the right menu.
7. Select the running application.
8. Refresh the browser watching the containers come and go with the demand.

Lab 3 - Utilize HDFS

This lab uses Hadoop's distributed file system, HDFS, to store and distributed files.

The goals of this lab are to:

- be able to start and stop HDFS
- create directories in HDFS
- copy files to HDFS
- use basic HDFS commands
- use the web application to view the filesystem and check HDFS's health

NOTE: If you receive the warning "WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable" it is just a warning and can be ignored. It is because Hadoop was compiled for 32-bit but we are running on a 64-bit system.

1. Configure the Hadoop core to know about HDFS by removing commented out XML.
   `$ sudo vim $HADOOP_HOME/etc/hadoop/core-site.xml`

```
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

2. Start HDFS.
   `$ start-dfs.sh`

3. Verify HDFS is running and that you see the NameNode, DataNode and SecondaryNameNode processes in addition to the YARN processes.
   `$ jps`
4. List the contents of the HDFS root.
   `$ hdfs dfs -ls /`
5. Make a directory called books.
   `$ hdfs dfs -mkdir /books`
6. List the contents of the HDFS root again to make sure you see the book directory.
   `$ hdfs dfs -ls /`
7. List the contents of the books directory.
   `$ hdfs dfs -ls /books`
8. Copy the Moby Dick book to the books directory on HDFS.
   `$ hdfs dfs -copyFromLocal /opt/data/moby_dick.txt /books`
9. Display the contents of Moby Dick from HDFS.
   `$ hdfs dfs -cat /books/moby_dick.txt`
10. Investigate some other HDFS commands.

- appendToFile
- cat
- chgrp
- chmod
- chown
- copyFromLocal
- copyToLocal
- count
- cp
- du
- get
- ls
- lsr
- mkdir
- moveFromLocal
- moveToLocal
- mv
- put
- rm
- rmr
- stat
- tail
- test
- text
- touchz

11. View the health of HDFS by opening a browser and navigate to http://localhost:50070/dfshealth.jsp.
12. Browser the filesystem and find Moby Dick.

Lab 4 - Combine Hadoop & HDFS

This lab will combine the distributed processing of Hadoop with the distributed filesystem of HDFS to create a distributed solution.

The goals of this lab are to:

- run a Hadoop job that uses data stored on HDFS
- view the results of a Hadoop job

1. Run the Hadoop wordcount example using the Moby Dick book as input and outputing the results to a new out directory.
   ```
   $ hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.2.0.jar wordcount /books out
   ```
2. Determine the files produced by the Hadoop job.
   ```
   $ hdfs dfs -ls out
   ```
3. View the contents of the _SUCCESS file.
   ```
   $ hdfs dfs -cat out/_SUCCESS
   ```
4. What is the purpose of the _SUCCESS file?
5. View the contents of the results file.
   ```
   $ hdfs dfs -cat out/part-r-00000
   ```
6. How many times was zeal found in Moby Dick?
7. Run the job again.
   ```
   $ hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.2.0.jar wordcount /books out
   ```
8. What was the output? Why was that the output?

Lab 5 - Write first Hadoop Map/Reduce Job

This lab includes writing your first mapper, reducer and job file and then runs them.

The goals of this lab are to:

- write a custom mapper
- write a custom reducer
- write a custom job

1. Change directory to the project workspace containing a project template.
   ```
   $ cd ~/workspaces/hadoop-wordcount
   ```

2. Create Mapper class.
   ```
   $ mkdir -p src/main/java/com/manifestcorp/hadoop/wc
   $ sudo vim src/main/java/com/manifestcorp/hadoop/wc/WordCountMapper.java
   ```

```
package com.manifestcorp.hadoop.wc;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class WordCountMapper extends Mapper<Object, Text, Text, IntWritable> {

    private static final String SPACE = " ";

    private static final IntWritable ONE = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context)
                        throws IOException, InterruptedException {
        String[] words = value.toString().split(SPACE);

        for (String str: words) {
            word.set(str);
            context.write(word, ONE);
        }

    }
}
```

3. Create Reducer class.
   ```
   $ sudo vim src/main/java/com/manifestcorp/hadoop/wc/WordCountReducer.java
   ```

```
package com.manifestcorp.hadoop.wc;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
```

```
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
                                            throws IOException, InterruptedException {
        int total = 0;

        for (IntWritable value : values) {
            total++;
        }

        context.write(key, new IntWritable(total));
    }
}
```

4. Create Job class.
   $ sudo vim src/main/java/com/manifestcorp/hadoop/wc/MyWordCount.java

```
package com.manifestcorp.hadoop.wc;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class MyWordCount {

    public static void main(String[] args) throws Exception {

        Job job = new Job();
        job.setJobName("my word count");
        job.setJarByClass(MyWordCount.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.setMapperClass(WordCountMapper.class);
        job.setReducerClass(WordCountReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

5. Build project.
   `$ mvn clean package`
6. Run job.
   `$ hadoop jar target/hadoop-mywordcount-0.0.1-SNAPSHOT.jar com.manifestcorp.hadoop.wc.MyWordCount /books out-2`
7. Determine the files produced by the Hadoop job.
   `$ hdfs dfs -ls out-2`
8. View the contents of the _SUCCESS file.
   `$ hdfs dfs -cat out-2/_SUCCESS`
9. What is the purpose of the _SUCCESS file?
10. View the contents of the results file.
    `$ hdfs dfs -cat out-2/part-r-00000`
11. How many times was zeal found in Moby Dick?

Appendix A - Environment Setup

This hands-on lab assumes you have virtual machine running Ubuntu 12.04 LTS and you have logged in with a user that has sudo privilages.

Tutorial Dependencies

1. Make directory to store dependencies.
   `$ sudo mkdir /opt/data`
2. Change directory to the newly created directory.
   `$ cd /opt/data`
3. Download Hadoop binary.
   `$ sudo wget http://www.trieuvan.com/apache/hadoop/common/hadoop-2.2.0/hadoop-2.2.0.tar.gz`
4. Optionally download Hadoop source code.
   `$ sudo wget http://www.trieuvan.com/apache/hadoop/common/hadoop-2.2.0/hadoop-2.2.0-src.tar.gz`
5. Download copy of Moby Dick and rename it.
   `$ sudo wget http://www.gutenberg.org/ebooks/2489.txt.utf-8`
   `$ mv 2489.txt.utf-8 moby_dick.txt`

Add Hadoop User and Group

1. Add hadoop group.
   `$ sudo addgroup hadoop`
2. Add hduser and put them in the hadoop group.
   `$ sudo adduser --ingroup hadoop hduser`

3. Give sudo rights to hduser.
   `$ sudo adduser hduser sudo`
4. Change user to hduser.
   `$ su hduser`
5. Change directory to the hduser's home directory.
   `$ cd ~`

Install Hadoop

1. Make a root directory for the Hadoop install.
   `$ sudo mkdir -p /opt/hadoop`
2. Uncompress the Hadoop tar file.
   `$ sudo tar vxzf /opt/data/hadoop-2.2.0.tar.gz -C /opt/hadoop`
3. Make the hduser the owner of the Hadoop home directory recursively.
   `$ sudo chown -R hduser:hadoop /opt/hadoop/hadoop-2.2.0`
4. Using a text editor, add environment variables to the end of hduser's .bashrc file.

```
# other stuff
# java variables
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64

# hadoop variables
export HADOOP_HOME=/opt/hadoop/hadoop-2.2.0
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
```

5. Apply the environment variables.
   `$ source .bashrc`
6. Make hadoop is installed and available.
   `$ hadoop version`

Configure YARN

1. From the /opt/hadoop/hadoop-2.2.0 directory, edit the YARN config file.
   `$ sudo vim 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>
```

2. Copy and edit a MapReduce config file.
   ```
   $ sudo mv etc/hadoop/mapred-site.xml.template etc/hadoop/mapred-site.xml
   $ sudo vim etc/hadoop/mapred-site.xml
   ```

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

3. As the hduser, configure a passwordless login.
   ```
   $ ssh-keygen -t rsa -P ”
   $ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
   ```
4. Validate the passwordless loging works.
   ```
   $ ssh localhost
   $ exit
   ```
5. Configure the JAVA_HOME environment variable.
   ```
   $ vim etc/hadoop/hadoop-env.sh
   ```

```
# other stuff
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
# more stuff
```

Configure HDFS

1. Make directory for the Name Node.
   ```
   $ sudo mkdir -p /opt/hdfs/namenode
   ```
2. Make directory for the Data Node.
   ```
   $ sudo mkdir -p /opt/hdfs/datanode
   ```

3. Update permissions on the previous directories to make them write able.
   ```
   $ sudo chmod -R 777 /opt/hdfs
   $ sudo chown -R hduser:hadoop /opt/hdfs
   ```
4. Update hdfs config file.
   ```
   $ cd /opt/hadoop/hadoop-2.2.0
   $ sudo vim etc/hadoop/hdfs-site.xml
   ```

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
  <property>
    <!-- number of hdfs instances to replicate to. -->
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <!-- location of Name Node data -->
    <name>dfs.namenode.name.dir</name>
    <value>file:/opt/hdfs/namenode</value>
  </property>
  <property>
    <!-- location of Data Node data -->
    <name>dfs.datanode.data.dir</name>
    <value>file:/opt/hdfs/datanode</value>
  </property>
</configuration>
```

5. Format HDFS.
   ```
   $ hdfs namenode -format
   ```
6. Configure the Hadoop core to know about HDFS.
   ```
   $ sudo vim etc/hadoop/core-site.xml
   ```

```
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

Git project templates

1. Change directory to the workspaces directory.
   ```
   $ cd ~/workspaces
   ```

2. Clone hadoop-logs project.
   ```
   $ git clone https://github.com/cjudd/hadoop-logs.git
   ```
3. Clone hadoop-wordcount project.
   ```
   $ git clone https://github.com/cjudd/hadoop-wordcount.git
   ```