**Hadoop Installation**

Note: These following instructions to install and run Hadoop are tested on lab machines.

Pre-req software:

sudo apt-get install -y ssh vim

sudo apt-get install -y pdsh

Download hadoop and unpack:

wget https://dlcdn.apache.org/hadoop/common/hadoop-3.3.1/hadoop-3.3.1.tar.gz

tar xzf hadoop-3.3.1.tar.gz

Now, you would see a folder named ‘hadoop-3.3.1’ in your directory

cd hadoop-3.3.1

Now, we need set the java\_home path in hadoop. To get your system’s java path, enter the following command:

dirname $(dirname $(readlink -f $(which javac)))

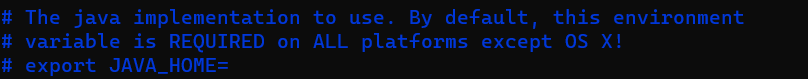
The output of the above command should be-

/usr/lib/jvm/java-8-openjdk-amd64

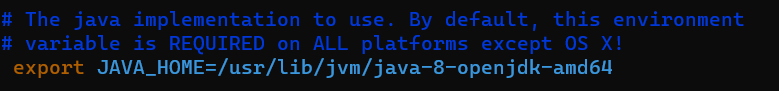
Copy the above output and open the following file-

vim etc/hadoop/hadoop-env.sh

After opening the above file, go to line 54 (you can set line number in vim editor by typing ‘:set number’), which should look like-



Now, uncomment the ‘export JAVA\_HOME=’ line by removing the starting ‘#’ symbol and paste the path of java over here. So, line 54 should look like-

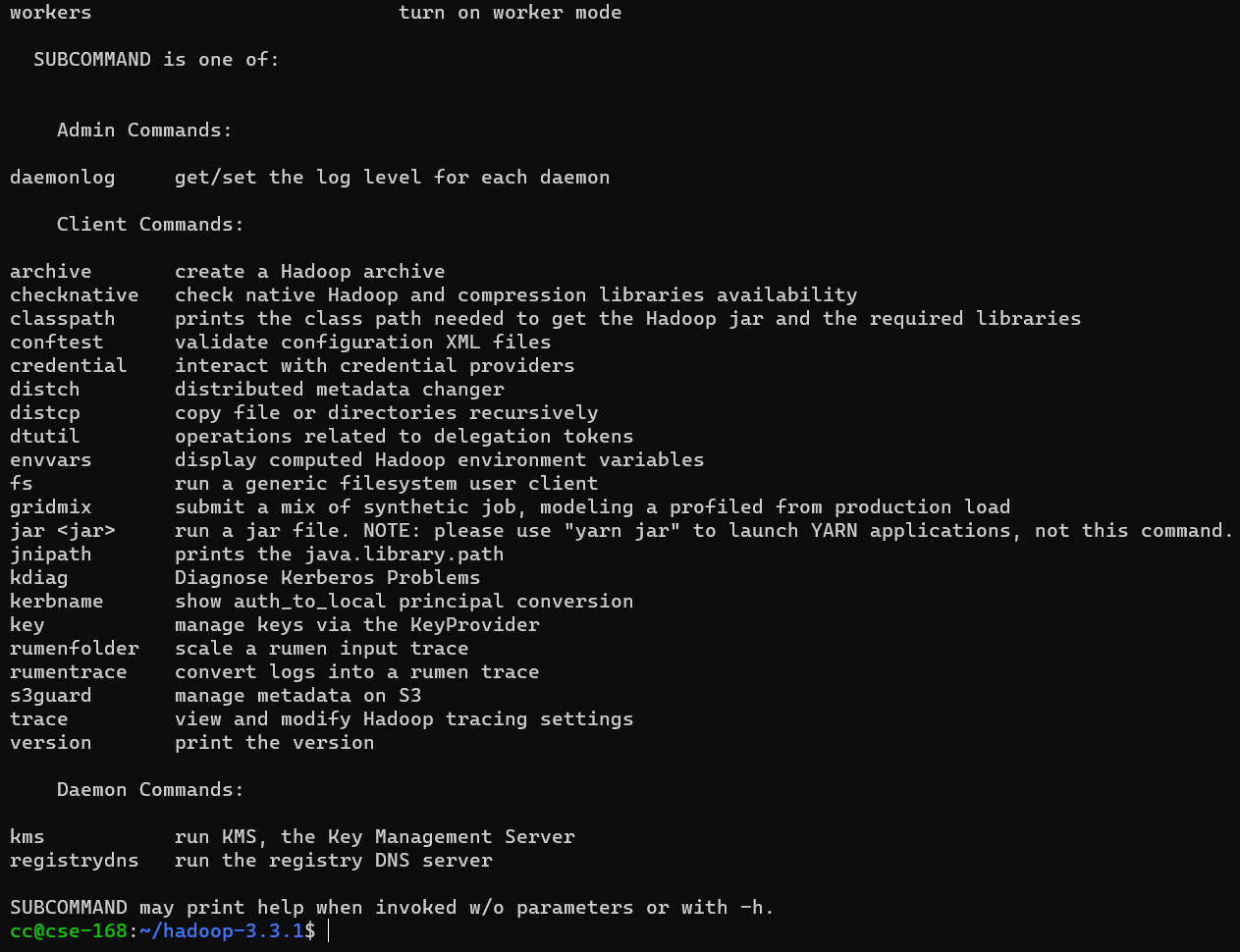


Now, save the file (press :w) and exit from vim editor (press :q).

Now, try the following command

bin/hadoop

You should the the usage documentation for the hadoop script as follows-



Now, we can run an existing example to test out running Hadoop cluster.

**Run Hadoop in standalone mode**

This means that Hadoop will run as a single Java process.

The following example copies the file in conf directory to use as input and then finds and displays every match of the given regular expression. Output is written to the given output directory.

mkdir input

cp etc/hadoop/\*.xml input

bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.1.jar grep input output 'dfs[a-z.]+'

cat output/\*

You should see the following output

1 dfsadmin

Let’s focus on the command *bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.1.jar grep input output 'dfs[a-z.]+'.* Here we pass a jar file (s*hare/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.1.jar)* along with the following arguments to hadoop-

1. Name of the program (‘grep’ in this case).
2. The input directory that contains the files which would be read by hadoop and in-turn passed as key-value pairs to the mapper function in the program (‘input’ directory in this case where we copies the xml file)
3. The output directory that will store the file containing the output (I.e., key-value pair) of reduce function in your program (‘output’ directory in this case).
4. (Optional) Any other arguments needed by your program (*'dfs[a-z.]+' in this case).*

You can learn more about jar here- <https://docs.oracle.com/javase/tutorial/deployment/jar/basicsindex.html>

Let’s view the output folder

ls output/ should the following two files-



And the contents of part-r-00000 should contain only one key-value pair matching the expression we specified-

1 dfsadmin

The above example asks hadoop to run a jar file containing the ‘grep’ program. So, how do we create a custom jar containing our program/logic/code and pass it on to hadoop to run?

**Creating a custom jar file**

For this, we will run WordCount (A map/reduce program that counts the words in the input files.) example on hadoop.

Run the following command to download the WordCount.java file

wget -v -O WordCount.java -L https://ucmerced.box.com/shared/static/7kmqxir7pqizx9jrmelzuuvmej6pqwam

Note: The WordCount.java code was taken from the following website- <https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#Example:_WordCount_v1.0>

*Please go through WordCount.java very very carefully as this will help you understand the syntax and flow of the mapper and reducer functions in this case. You can take WordCount.java as your base file and modify it to finish any one of the Lab-2 Map/Reduce programs.*

Now, let’s try to make a jar for WordCount. The first step is to compile your program-

mkdir WordCount

javac -classpath $(bin/hadoop classpath) -d WordCount/ WordCount.java

If the above command runs without any errors then your compilation was a success. You can also view the contents in the WordCount/ folder which will contain the class files-

ls WordCount/

'WordCount$IntSumReducer.class' 'WordCount$TokenizerMapper.class' WordCount.class

To make the jar run the following command-

jar -cvf WordCount.jar -C WordCount/ .

When you list the directory (ls) you should be able see a jar file, namely WordCount.jar being created. Now, we can pass this jar to hadoop to get word count of a file. But before we do that, we need to create an input file. You can do it as follows:

1. First, we need to create an input directory-

mkdir wc-input

1. Create a file named ‘file0’ in the input directory (the filename does not matter)-

vim wc-input/file0

1. Enter the following contents in the file (you can enter other words as well)-

Hello World Bye World

Hello Hadoop Goodbye Hadoop

1. Now, save the file and exit vim.

Now, you can run the word count program as follows-

bin/hadoop jar WordCount.jar WordCount wc-input/ wc-output

Once the program ends, you can view the output as follows-

cat wc-output/part-r-00000

And it should look like-

Bye 1

Goodbye 1

Hadoop 2

Hello 2

World 2

You can follow similar steps to run your own programs on hadoop. Remember to carefully understand the word count example as it will help you to build your map/reduce own programs. You can find detailed explanation of the code here- <https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#Walk-through>

Few things to remember while running hadoop-

1. When you re-run a program, say word count, you need to remove the output directory before running it again. For example, in this case you need to remove the ‘wc-output’ directory as follows-

rm -rf wc-output/