Installing and Running Hadoop and Spark on Ubuntu 18

1. Installing Java for Hadoop

Hadoop requires Java to be installed, so check first if ubuntu is pre occupied with java by default.

```
$ java -version
```

If java is not preinstalled in my system, so i installed java with SKDMAN. Using SKDMAN we can handle the different versions of java very efficiently. Hadoop runs smoothly with Java 8, but may encounter bugs with newer versions of Java. So I'd like to install Java 8 specifically.

```
$ sudo apt install curl -y
$ curl -s "https://get.sdkman.io" | bash
```

To see the available softwares on SKDMAN package use:

```
$ sdk ls
```

To make sure you can use SDKMAN! in every new terminal, run the following command to append a line which sources the SDKMAN! initialisation script whenever a new terminal is opened:

```
$ echo "source ~/.sdkman/bin/sdkman-init.sh" >> ~/.bashrc
```

To list the all available version in java use :

```
$ sdk list java
```

To is tall a specific java version:

```
$ sdk install <software> <Identifier>
$ sdk install java 8.0.232.hs-adpt
```

To check the installed version of java is SDKMAN

```
$ ls ~/.sdkman/candidates/java
```

To verity the java version use:

```
$ java -version
```

If you install multiple Java versions, you can easily switch between them using following SDKMAN code.

```
$ sdk install java 13.0.1.hs-adpt
$ sdk use java 13.0.1.hs-adpt
```

We also need to explicitly define the JAVA_HOME environment variable by adding it to the ~/.bashrc file:

```
$ echo "export JAVA_HOME=\$(readlink -f \$(which java) | sed 's:bin/java::')" >> ~/.bashrc
$ echo $JAVA_HOME     # to verifying it
```

To chage the java version back to 8, for continue with hadoop instalation use:

```
$ sdk use java 8.0.232.hs-adpt
```

2. Installing Hadoop

Install the most recent and stable version of Hadoop from :
http://mirrors.whoishostingthis.com/apache/hadoop/common/hadoop-3.2.1/hadoop-3.2.1-src.tar.gz.

We can either download it using the browser directly or download it using the terminal.

```
$ wget http://mirrors.whoishostingthis.com/apache/hadoop/common/hadoop-3.2.1/hadoop-3.2.1-src.tar.gz.
```

Unpack the archive with tar, and redirect the output to the /opt/ directory. Then Remove the archive file and move to the /opt/ directory.

```
$ sudo tar -xvf hadoop-3.2.1.tar.gz -C /opt/
$ rm hadoop-3.2.1.tar.gz && cd /opt
```

Rename the Hadoop directory to hadoop and change its permissions so that its owned by me and not by root:

```
$ sudo mv hadoop-3.2.1 hadoop && sudo chown shyam:shyam -R hadoop
```

Define the HADOOP_HOME environment variable and add the correct Hadoop binaries to my PATH by echoing the following lines and concatenating them to my ~/.bashrc file:

```
$ echo "export HADOOP_HOME=/opt/hadoop" >> ~/.bashrc
$ echo "export PATH=\$PATH:\$HADOOP_HOME/bin:\$HADOOP_HOME/sbin" >> ~/.bashrc
```

To verify the hadoop instalation use :

```
$ hadoop version
```

In order for HDFS to run correctly later, we also need to define JAVA_HOME in the file /opt/hadoop/etc/hadoop/hadoop-env.sh. So fine the file hadoop-env.sh in /opt/hadoop/etc/hadoop/.
Open it using any text or code editor - (I used Vim) and fine the line - # export JAVA_HOME=.
Remove the # symbol and add the below line and save it.

```
export JAVA_HOME=/home/shyam/.sdkman/candidates/java/8.0.232.hs-adpt
```

3. Installing Spark

Download the latest version of apache spark from the spark website : https://spark.apache.org/

I downloaded and installed the Spark 3.0.0-preview2 (23 Dec 2019) pre-built for Apache Hadoop 3.2 and later with the following command.

We can download the above spark version using browser or terminal,following the given link : http://ftp.heanet.ie/mirrors/www.apache.org/dist/spark/spark-3.0.0-preview2/spark-3.0.0-preview2-bin-hadoop3.2.tgz

```
$ wget http://ftp.heanet.ie/mirrors/www.apache.org/dist/spark/spark-3.0.0-preview2/spark-3.0.0-preview2-bin-hadoop3.2.tgz
```

Unpack the archive with tar, and redirect the output to the /opt/ directory. Then remove the archive file and move to the /opt/ directory. Rename the Spark directory and change its permissions, so its owned by you (shyam) and not root.

```
$ sudo tar -xvf spark-3.0.0-preview-bin-hadoop3.2.tgz -C /opt/
$ rm spark-3.0.0-preview-bin-hadoop3.2.tgz && cd /opt
$ sudo mv spark-3.0.0-preview-bin-hadoop3.2 spark && sudo chown shyam:shyam -R spark
```

Finally, define the SPARK_HOME environment variable and add the correct Spark binaries to your PATH by echoing the following lines and concatenating them to your ~/.bashrc file:

```
$ echo "export SPARK_HOME=/opt/spark" >> ~/.bashrc
$ echo "export PATH=\$PATH:\$SPARK_HOME/bin" >> ~/.bashrc
```

Verify the instalation using the following line of code.

```
$ spark-shell –version
```

4.Configuring HDFS

At this point, Hadoop and Spark are installed and running correctly, but we haven't yet set up the Hadoop Distributed File System (HDFS).

To configure HDFS, we need to edit several files located at : `/opt/hadoop/etc/hadoop/`

Chage the following files as follows.

a. core-site.xml

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

b. hdfs-site.xml

```
<configuration>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:///opt/hadoop_tmp/hdfs/datanode</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:///opt/hadoop_tmp/hdfs/namenode</value>
  </property>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
</configuration>
```

We set dfs.replication to 1 because we can only create a single cluster in a local machine.

The directories given above (/opt/hadoop_tmp/hdfs/datanode and /opt/hadoop_tmp/hdfs/namenode) must exist and be read/write-able by the current user. So create them now, and adjust their permissions, with:

```
$ sudo mkdir -p /opt/hadoop_tmp/hdfs/datanode
$ sudo mkdir -p /opt/hadoop_tmp/hdfs/namenode
$ sudo chown shyam:shyam -R /opt/hadoop_tmp
```

c. mapred-site.xml

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

d. yarn-site.xml

```
<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
  <property>
    <name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>
  </property>
</configuration>
```

5. Configuring SSH

If you started with a minimal Ubuntu installation like I did, you may need to first set up your ssh connection (as HDFS connects to localhost:9000). To check if the SSH server is running, enter the command.

```
$ which sshd
```

If nothing is returned, then the SSH server is not installed (this is the case with the minimal Ubuntu installation). To get this up and running, install openssh-server, which will start the SSH service automatically:

```
$ sudo apt install openssh-server
$ sudo systemctl status ssh  # to check the status of the SSH install.
```

To check that this worked, try ssh-ing into localhost:

```
$ ssh localhost     # give yes to continue with the instalation. We can use exit to end the connection.
```

Then, create a public-private keypair (if you haven't already):

```
$ ssh-keygen
```

Hit 'enter' / 'return' over and over to create a key in the default location with no passphrase. When you're back to the normal shell prompt, append the public key to your ~/.ssh/authorized_keys file:

```
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

Now we should be able to boot HDFS.

6. Formatting and Booting HDFS

At this point, we can format the distributed filesystem.
Format the HDFS with

```
$ hdfs namenode -format -force
```

We can then boot the HDFS with the following two commands:

```
$ start-dfs.sh && start-yarn.sh
```

You can check that HDFS is running correctly with the command jps

```
$ jps
```

#output

```
21841 SecondaryNameNode
21618 DataNode
22322 NodeManager
22679 Jps
22152 ResourceManager
21453 NameNode              # You should see a NameNode and a DataNode, at minimum, in that list.
```

Check that HDFS is behaving correctly by trying to create a directory, then listing the contents of the HDFS:

```
$ hdfs dfs -mkdir /test
$ hdfs dfs -ls /
```

If you can see your directory, you've correctly configured the HDFS

Hadoop and Spark come with built-in web-based monitors that you can access by going to:
http://localhost:8088
http://localhost:9870

7. Checking with Spark and HDFS

One of the benefits of working with Spark and Hadoop is that they're both Apache products, so they work very nicely with each other. It's easy to read a file from HDFS into Spark to analyse it. To test this, let's copy a small file to HDFS and analyse it with Spark.

Spark comes with some example resource files. With the above configuration, they can be found at /opt/spark/examples/src/main/resources. Let's copy the file users.parquet to HDFS:

```
$ hdfs dfs -put /opt/spark/examples/src/main/resources/users.parquet /users.parquet
```

Next, open the Spark shell and read in the file with read.parquet:

```
$ spark-shell  # to open the spark shell.
```

Read the file in HDFS using scala in spark.

```
scala> val df = spark.read.parquet("hdfs://localhost:9000/users.parquet")
df: org.apache.spark.sql.DataFrame = [name: string, favorite_color: string ... 1 more field]
scala> df.collect.foreach(println)
```

If you want to stop the HDFS, you can run the commands:

```
$ stop-dfs.sh
$ stop-yarn.sh
```

* this document is only created for project documentation purpose and referencing only. All the steps followed in this document is complety based on blog written by Andrew . Link to the blog: https://dev.to/awwsmm/installing-and-running-hadoop-and-spark-on-ubuntu-18-393h