Creating a Virtual Environment for Spark

Most of the computing on data-centers happens in virtualized environments. If we want to run our Hadoop + Spark in an isolated environment, ready exclusively for our analytics without modifying our base system, we can create a virtual machine (VM) to install and run everything inside. Here we will create a VM, and deploy an Ubuntu OS inside where we'll deploy Spark and its required tools.

This first step will install the VM environment, where we are deploying Spark and the Hadoop Distributed FileSystem (HDFS).

Vagrant and VirtualBox

First, we need to install VirtualBox (virtualizer) and Vagrant (manager of VM images).

If we are running Ubuntu/Debian, we can run the following command. Follow the instructions and accept the dependencies of those packages. You may need to restart your system if required by the system.

```
sudo apt-get install virtualbox vagrant
```

If you are running MacOS, Windows, or you have problems with Linux, check this URL for instructions to install VirtualBox https://www.virtualbox.org/wiki/Downloads and Vagrant https://www.vagrantup.com/downloads.html

Ubuntu VM

Once we have VirtualBox and Vagrant installed, we can download a pre-installed VM with Ubuntu. The following commands work either for Linux, MacOS and Windows command line:

```
mkdir vm_test                     # Create a working directory
cd vm_test                        # Enter the directory
vagrant init ubuntu/bionic64      # Download the "VM descriptor"
```

Now we modify the "VM descriptor", to tell the VM to use 4GB of RAM. Edit the file "Vagrantfile", and add the following lines in the configuration, in between the "config" tags:

```
 config.vm.provider "virtualbox" do |vb|
       vb.memory = "4096"
 end
```

Now, our VM is ready for being downloaded (the first time) and started, using the command:

```
vagrant up                # Download (1st time) and Start the VM
```

Entering in our VM

The VM should have started, and we can enter doing:

```
vagrant ssh               # Opening a Secure-Shell session
```

...or in case you decided to start the VM using the “VirtualBox GUI”, you can start it from there.

Now we are in the VM and we can install the things we need. To exit the VM just type “exit”, to stop the VM use the command “vagrant halt”, and to start the VM again use the command “vagrant up”. To delete the VM use “vagrant destroy”, and when doing “vagrant up” again, the VM will be downloaded as new.

From here on **WE ARE DOING EVERYTHING INSIDE THE VM**.

Installing HDFS – Hadoop

Prerequisites

Installing Java 8

Hadoop and Spark require a working Java installation. We are using version Java 1.8, as it is known to be fully compatible:

```
sudo apt-get update
sudo apt-get install openjdk-8-jdk
cd /usr/lib/jvm
sudo ln -s java-8-openjdk-amd64 default-java
```

After installation, make a quick check if Java is correctly set up:

```
java -version
```

And we should see something like that, and be sure version is +1.8:

```
openjdk version "1.8.0_275"
OpenJDK Runtime Environment (build 1.8.0_275-8u275-b01-0ubuntu1~18.04-b01)
OpenJDK 64-Bit Server VM (build 25.275-b01, mixed mode)
```

Adding a dedicated User for HDFS

Now we create the UNIX group “hadoop”, and we add our user to that group. This will create the group and add the user that will run HDFS and Spark to the group hadoop (in case of the VM, the user is “vagrant”).

```
sudo addgroup hadoop
sudo usermod -a -G hadoop vagrant
```

Configuring SSH

HDFS requires SSH access to manage its nodes. For our single-node setup of Hadoop, we therefore need to configure SSH access to “localhost” for our user. Now, we need to be able to open SSH connections without password, using RSA keys. First, we have to generate an SSH key for the user (leave the saving path as default):

```
ssh-keygen -t rsa -P ""
```

Follow the instructions and you should have the files **~/.ssh/id_rsa** and **~/.ssh/id_rsa.pub** created. Next, you have to enable SSH access to your local machine with this newly created key:

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

This will add your RSA public key to your list of allowed keys, so you can SSH your localhost without requesting password. Let's check if it worked:

```
ssh localhost
```

The first time you connect to a machine through SSH, it will ask you to trust the remote machine's "fingerprint" and save it into your user's known_hosts file (only the first time you connect). By testing the connection we're completing this step, so don't forget to test every connection to any machine before adding it to your Hadoop deployment. **After connecting to localhost, exit this loopback connection using "exit"**.

Installing Hadoop

Installation

Download Hadoop (using the shortcut here used) and extract it to a location of your choice (e.g. /opt/hadoop) as follow. Make sure to change the owner of all the files to the user ("vagrant" in the VM) and hadoop group.

```
cd /opt/
sudo wget https://bit.ly/4bklzEd -O hadoop-3.3.0.tar.gz
sudo tar xvzf hadoop-3.3.0.tar.gz
sudo ln -s hadoop-3.3.0 hadoop
sudo chown -R vagrant:hadoop hadoop-3.3.0 hadoop
```

Update your .bashrc

Add the following lines to the end of **~/.bashrc** file of the user, or the RC file of your current shell:

```
# Set Hadoop-related environment variables
export HADOOP_HOME=/opt/hadoop

# Set JAVA_HOME (we will also configure it later on Hadoop)
export JAVA_HOME=/usr/lib/jvm/default-java

# Add Hadoop bin/ directory to PATH
export PATH=$PATH:$HADOOP_HOME/bin
```

After this, we can reload ~/.bashrc and check the configuration:

```
source ~/.bashrc
echo $JAVA_HOME
echo $HADOOP_HOME
```

Configuration

This configuration targets to set-up a single Hadoop node. Check out the Hadoop Wiki for more information about tuning the configuration for multi-node clusters.

Environment

$HADOOP_HOME/etc/hadoop/hadoop-env.sh

The only required environment variable we have to configure for Hadoop in this tutorial is JAVA_HOME. Open $HADOOP_HOME/etc/hadoop/hadoop-env.sh in the editor of your choice and set the JAVA_HOME environment variable to the OpenJDK 8 directory.

```
# The java implementation to use.  Required.
export JAVA_HOME=/usr/lib/jvm/default-java

# disable ipv6 for Hadoop
export HADOOP_OPTS=-Djava.net.preferIPv4Stack=true
```

Configuration Files

In this section, we will configure the directory where Hadoop will store its data files, the network ports it listens to, etc. Our setup will use Hadoop's Distributed File System, HDFS, even though our "cluster" only contains our single local machine.

You can leave the settings below "as is" with the exception of the hadoop.tmp.dir parameter – this parameter you must change to a directory of your choice. We will use the directory /app/hadoop/tmp in this tutorial. Hadoop's default configurations use hadoop.tmp.dir as the base temporary directory both for the local file system and HDFS, so don't be surprised if you see Hadoop creating the specified directory automatically on HDFS at some later point.

HDFS Directory

Now we create the directory and set the required ownership and permissions for our user (in the VM, this is "vagrant"):

```
sudo mkdir -p /app/hadoop/tmp
sudo chown vagrant:hadoop /app/hadoop/tmp
sudo chmod 750 /app/hadoop/tmp
```

If you forget to set the required ownership and permissions, you will see a java.io.IOException when you try to format the name node in the next section.

Add the following snippets **between the <configuration> ... </configuration> tags** in each of the following configuration XML files:

*$HADOOP_HOME/etc/hadoop/*core-site.xml

```
<property>
  <name>hadoop.tmp.dir</name>
  <value>/app/hadoop/tmp</value>
  <description>A base for other temporary
directories.</description>
</property>

<property>
  <name>fs.default.name</name>
  <value>hdfs://localhost:54310</value>
  <description>Location for the default file
system.</description>
</property>
```

*$HADOOP_HOME/etc/hadoop/*mapred-site.xml

```
<property>
  <name>mapred.job.tracker</name>
  <value>localhost:54311</value>
  <description>The host and port that the MapReduce job tracker
runs at.</description>
</property>
```

*$HADOOP_HOME/etc/hadoop/*hdfs-site.xml

```
<property>
  <name>dfs.replication</name>
  <value>1</value>
  <description>Default block replication.</description>
</property>
```

Formatting the HDFS filesystem via the NameNode

The first step to starting up your Hadoop installation is formatting the Hadoop file-system which is implemented on top of the local file-system of your "cluster" (which includes only your local machine if you followed this tutorial). You need to do this the first time you set up a Hadoop cluster. Do not format a running Hadoop file-system as you will lose all the data currently in the HDFS cluster!

To format the File System (which simply initializes the directory specified by the fs.name.dir variable), run the command:

```
 $HADOOP_HOME/bin/hadoop namenode -format
```

Starting HDFS

Once the system is formatted, you can start HDFS using the following command:

```
 $HADOOP_HOME/sbin/start-dfs.sh
```

This will start a Namenode, Datanode, Jobtracker and a Tasktracker on your machine. If no error messages appeared, you should see your new HDFS system by listing it using "ls". You can test creating directories and putting files using the "hdfs dfs" command, found in $HADOOP_HOME/bin (now added to your $PATH if followed the previous steps).

```
 cd                                          # return to "Home"
 touch example_file.txt                     # Create local file
 hdfs dfs -mkdir /test_dir                  # Create a directory
 hdfs dfs -put example_file.txt /test_dir/  # Upload the file
 hdfs dfs -ls /test_dir                     # Show the file
```

Remember that HDFS has also user permissions, and you can use the "chmod" command in "hdfs dfs" to change HDFS permissions for files and directories.

Now, just remember which port we configured for HDFS, as we put in the "core_site.xml" file (here *hdfs://localhost:54310*). We will use this for connecting Spark to the HDFS and read its files. By running netstat we can see which ports is JAVA using, and we will see the one indicated at Hadoop's "fs.default.name" configuration.

```
sudo netstat -plten | grep java
```

We should see a java service running on "54310":

```
tcp    0    0    127.0.0.1:54310    0.0.0.0:*    LISTEN    1000    6090571    22603/java
…
```

Stopping HDFS

To stop the HDFS we can run the command stop-dfs.sh as shown:

```
$HADOOP_HOME/sbin/stop-dfs.sh
```

By running start-dfs.sh again, we will resume the HDFS execution, where data is persistent.

Installing Apache Spark

Now we'll download and prepare SPARK 3.5 (for Hadoop3.3), then set-up the environment, always inside the Virtual Machine.

Download Spark:

```
Wget https://bit.ly/420oFcD -O spark-3.5.0-bin-hadoop3.tgz
tar xvzf spark-3.5.0-bin-hadoop3.tgz
ln -s spark-3.5.0-bin-hadoop3 spark
```

Then set-up the environment:

```
Edit ~/.bashrc and add the lines:

# Set Spark-related environment variables
export SPARK_HOME=/home/vagrant/spark
export PATH=$PATH:$SPARK_HOME/bin
```

Finally, reload the environment:

```
source ~/.bashrc
```

Installing R-cran base for SparkR

If we want to test SparkR, we need to install the base for R-cran:

```
sudo apt-get update
sudo apt-get install r-base
```

Other Course Preparations

During the course, we will use some big data-sets, obtained from the Internet. Better download them now to avoid waiting in the course while files are downloaded. Do it **inside the VM**.

Download the Data-Sets:

```
wget http://bit.ly/1Aoywaq --output-document=donation.zip
wget http://bit.ly/2jZgeZY --output-document=csv_hus.zip
```

Install additional programs (unzip):

```
sudo apt-get install unzip
```

Summary of VM and HDFS commands

Vagrant commands for a "ubuntu/bionic64" VM. Remember that your "VM Descriptor" is called "Vagranfile", you can modify it to give the VM more memory, open ports or give SSH passwords.

```
vagrant init ubuntu/bionib64    # Download the "VM descriptor"

vagrant up      # start the VM (it downloads it the 1st time)
vagrant halt    # stop the VM
vagrant ssh     # open an SSH connection with the VM
vagrant destroy # delete de VM (you can download later again)
```

Commands to connect to the VM from alternative methods (not vagrant). If you are running the VM from a VirtualBox GUI, you can open a SSH connection from another terminal by:

1) Discovering your VM vagrant SSH key and look for the "IdentityFile", something like "`IdentityFile /home/.../private_key`", and the "Port", e.g. "`Port 2222`":

```
vagrant ssh-config
```

2) Use that key and port to open a SSH connection with your SSH client. E.g. openssh-client:

```
ssh -p 2222 vagrant@127.0.0.1 -o IdentityFile=/home/.../private_key
```

Hadoop Distributed File System (HDFS) commands. Remember that usual SH commands work inside the HDFS, like "chmod", "mv", "cp", ...

```
$HADOOP_HOME/sbin/start-dfs.sh    # Start the HDFS
$HADOOP_HOME/sbin/stop-dfs.sh     # Stop the HDFS

hdfs dfs -ls /              # List the directory "/"
hdfs dfs -mkdir /DIR1       # Creates DIR1 at "/"
hdfs dfs -put FILE1 /       # Insert FILE1 into HDFS at "/"
hdfs dfs -get /FILE1 ./     # Recover FILE1 into your local FS

$HADOOP_HOME/bin/hadoop namenode -format   # Format the HDFS
```