Lab 2. HDFS

Exercise1. Create a single node HDFS.

Step 1. Install the HDFS Namenode.

Run the following command:

```
yum install -y hadoop-hdfs-namenode
```

Once installed, you can see all possible options for the namenode service by running the following:

```
service hadoop-hdfs-namenode
```

Note that hadoop installs its own user and group accounts as well as installing its base directory in /usr/lib/hadoop.

The configuration directory for hadoop is in /etc/hadoop/conf. All the configuration files are located there.

The environment variables that are set for Hadoop are located in /etc/default/hadoop.

Now install the 'pseudo' cluster configuration for a single node HDFS instance.

```
yum install -y hadoop-conf-pseudo
rpm -ql hadoop-conf-pseudo
```

The output should be a list of the directories and files installed by yum for the pseudo package.

Much of hadoop is configured via XML documents. An easy way to find out the configuration value for any hadoop parameter is to create a script like so:

```
#!/bin/sh
echo '<property>';sed -N ">$1</,/property/p/" $2
```

You can then test it by simply running it like so: <scriptname> dfs.namenode.name.dir hdfs-default.xml

This will print out the value of dfs.namenode.name.dir in the hdfs-default.xml file (which we don't have yet).

Next, format the HDFS file system. Note that you must su to the hdfs user to do this.

```
su –c 'hdfs namenode –format' hdfs
```

Now you can start the hdfs name service.

```
service hadoop-hdfs-namenode start
service hadoop-hdfs-namenode status
```

If everything is installed correctly, then you should get output that looks like this:

```
Hadoop namenode is running
[ OK ]
```

Step 2. View HDFS parameters and important files.

The lab systems may be limited in the amount of RAM. We can decrease the amount of RAM used by the name node.

```
pmap –d $(pgrep –f namenode)
```

This will give us the amount of RAM used by the namenode process.

Then we decrease it by setting the hadoop configuration parameter HADOOP_HEAPSIZE in the /etc/hadoop/conf/hadoop-env.sh file to the value of 200.

Now we can restart the namenode using the service restart command.

Next, we can look at the status of the hdfs system by using the hdfs dfsadmin command.

```
su –c 'hdfs dfsadmin –report' hdfs
```

Initially, the values it shows should be zero as we've not added any hdfs nodes to the file system, so let's add the current node to the HDFS file system like so:

```
service hadoop-hdfs-datanode start
jps
```

Jps is one of the JDK tools we installed in the last lab. Since hadoop runs inside the JVM, we need to find out which hadoop processes are running inside the JVM.

Running the hdfs dfsadmin –report command again now gives us some non-zero values.

We can also take a look at the files that store the hdfs metadata.

```
su -c "hdfs getconf -confkey dfs.namenode.name.dir" hdfs
```

This first command tells us where the directory that stores the name metadata is located.

Use the *ls* command to list all of the files in that directory. The *current* subdirectory contains the relevant files.

Use the *cat* command to look at the contents of the *VERSION* file.

Let's next examine the files that contain the hdfs backing store. The backing store is the area that actually stores the data files for the hdfs file system.

As before, use the hdfs getconf –getconfkey with the *dfs.datanode.data.dir* parameter to obtain the location of the backing store.

Then, go into the *current* directory and cat the *VERSION* file.

Let's examine the permissions and ownership settings since the last restart.

```
egrep -A2 'fsOwner' /var/log/hadoop-hdfs/hadoop-hdfs-namenode*log |
tail -n3
```

The amount of available space and permissions and ownership of the root of the HDFS file system can be viewed via this command:

```
hdfs dfs –df
hdfs dfs -ls -d /
```

Note that the owner of the root hdfs system is the hdfs user. Not the root user. Attempting to copy a file from the local file system to the hdfs file system will give a permission denied error.

For example, run the following command to attempt to copy the local /var/log/messages file to the root of the HDFS.

```
hdfs dfs -put /var/log/messages /
```

Additionally, attempting to copy the /var/log/messages file to the HDFS file system as the hdfs user will also result in an error:

```
su –c ‘ hdfs dfs -put /var/log/messages /’ hdfs
```

This is because the hdfs user does not have access permissions to the /var/log/messages file.

In order to facilitate this, upload the file by reading it as the local root user and writing it to the HDFS as the hdfs user like so:

```
cat /var/log/messages | su -c "hdfs dfs -put - /messages" hdfs
hdfs dfs –ls /
```

HDFS does not employ the concept of a current working directory. Any non-absolute paths are interpreted as being relative to the users home directory at /user/username.

For example, do the following:

```
cat /var/log/messages | su -c "hdfs dfs -put - messages" hdfs
```

Note that when you do an hdfs dfs –ls /user/hdfs you will see a file called *messages*.

We can also copy a file from the hdfs file system to the local file system by using the –get option to the hdfs dfs command. Please do the following:

```
hdfs dfs -get /messages /tmp
```

Now running ls on the /tmp directory should show you the messages file stored in the local /tmp directory.

Run the –help option to hdfs dfs to see all of the other options available to you via hdfs

```
hdfs dfs -help | less
```

Step 4. Cleaning up.

We are now going to clean up the node so that we can do the next lab, which will be to install HDFS in multi-node configuration.

First, save our configuration file. We'll need this later.

```
cp /etc/hadoop/conf/hadoop-env.sh /root
```

Next, delete the extra hdfs packages that needed the main packages as dependencies by running the following:

```
# yum remove hadoop-hdfs-secondarynamenode hadoop-yarn-
{node,resource}manager hadoop-mapreduce-historyserver
```

The only thing that should be left are the namenode and datanode daemons. Verify that they are still there by running the following command:

```
chkconfig --list | grep hadoop
```

Next, stop all of the services via the service stop command.

```
service hadoop-hdfs-datanode stop
service hadoop-hdfs-namenode stop
```

Finally, remove the data and meta-data from the hdfs system

```
rm -rf /var/lib/hadoop-hdfs/cache/hdfs/dfs/*
```

Exercise2. Create a multi-node HDFS.

Step 1. Install the HDFS base and datanode packages on all your cluster nodes.

```
allnodes 'yum install -y hadoop-hdfs hadoop-hdfs-datanode'
```

Step 2. Configure storage space for the HDFS

We'll create a thinly provisioned LVM logical volume on each node. Note that in the real world, it is more likely to create physical volumes, but for our lab environment, we'll use LV's. Verify that each LV was created on each node.

```
allnodes lvcreate -L 10G -thinpool vg0/hdfs-pool
for i in 1 2 3 4; do allnodes "lvcreate -n hdfs$i -V 5G --thinpool
```

```
vg0/hdfs-pool"; done
allnodes -i "lvs | grep hdfs"
```

Now we'll create file systems on each of our new block devices. Note here that we're using the XFS file system, however EXT3 and EXT4 are also perfectly acceptable alternatives.

```
For I in 1 2 3 4: do allnodes 'mkfs -t xfs /dev/vg0/hdfs$i
```

Now we need to add each file system to the /etc/fstab directory on each node. Add the following lines to the fstab file.

```
/dev/vg0/hdfs1 /var/lib/hadoop-hdfs/data1 xfs noatime, nodiratime 1 2
/dev/vg0/hdfs2 /var/lib/hadoop-hdfs/data2 xfs noatime, nodiratime 1 2
/dev/vg0/hdfs3 /var/lib/hadoop-hdfs/data3 xfs noatime, nodiratime 1 2
/dev/vg0/hdfs4 /var/lib/hadoop-hdfs/data4 xfs noatime, nodiratime 1 2
```

Copy this new fstab to all the other datanodes in the cluster.

```
pscp.pssh -H 'node2 node3 node4' /etc/fstab /etc
```

You will also need to set mount points and permissions for each file system.

```
mkdir -p /var/lib/hadoop-hdfs/{cache,data{1,2,3,4}}
mount -a
mount | grep hdfs
chown hdfs:hadoop /var/lib/hadoop-hdfs/data{1,2,3,4}
chmod 755 /var/lib/hadoop-hdfs/data{1,2,3,4]
ls -ld /var/lib/hadoop-hdfs/data{1,2,3,4}
```

Now we need to create a new directory for the Namenode so that it can store the HDFS meta-data and set permissions appropriately.

```
mkdir -m 700 /var/lib/hadoop-hdfs/name
chown hdfs:hadoop /var/lib/hadoop-hdfs/name
```

Step 3. Configure XML configuration parameters for the HDFS

On each node run the following commands to create and set a new config directory.

```
cp -a /etc/hadoop/conf.empty /etc/hadoop/conf.4node
```

```
alternatives –install /etc/hadoop/conf hadoop-conf
/etc/hadoop/conf.4node/ 20
alternatives –display hadoop-conf
```

On the first node, node1, edit the core-site.xml config file and define the following property underneath the <configuration> tag.

```
<property>
<name>fs.defaultFS</name>
<value>hdfs://node1:8020</value>
</property>
```

The Fs.defaultFS property is used by the namenode to determine what TCP port to listen on. It's also used by the other nodes to determine where to find the primary namenode.

Now edit the hdfs-site.xml config file and define the following properties underneath the <configuration> tag.

```
<property>
<name>dfs.namenode.name.dir</name>
<value>file:///var/lib/hadoop-hdfs/name</value>
</property>
<property>
<name>dfs.namenode.checkpoint.dir</name>
<value>file:///var/lib/hadoop-hdfs/namesecondary</value>
</property>
<property>
<name>dfs.datanode.data.dir</name>
<value>file:///var/lib/hadoop-hdfs/data1,file:///var/lib/hadoop-
hdfs/data2,
file:///var/lib/hadoop-hdfs/data3,file:///var/lib/hadoop-
hdfs/data4</value>
</property>
<property>
<name>hadoop.tmp.dir</name>
<value>file:///var/lib/hadoop-hdfs/cache/</value>
</property>
<property>
<name>dfs.blocksize</name>
<value>10M</value>
</property>
```

Note that the value for the dfs.datanode.datadir is one line.

Now restore our configuration file that we saved from the last lab.

```
cp /root/hadoop-env.sh /etc/hadoop/conf.4node
chmod 755 /etc/hadoop/conf/hadoop-env.sh
```

Now we will push the config files out to all of the cluster nodes.

```
cd /etc/hadoop/conf.4node/
for i in 2 3 4; do scp core-site.xml hdfs-site.xml hadoop-env.sh
node$i:/etc/hadoop/conf/; done
```

Step 4. Format the HDFS and start the services.

As with the last lab, we'll need to format the hdfs file system.

```
su –c 'hdfs namenode –format' hdfs
```

Now run the namenode.

```
service hadoop-hdfs-namenode start
service hadoop-hdfs-namenode status
```

Next, we'll start the datanode service on all of the data nodes.

```
pssh –H 'node1 node2 node3' service hadoop-hdfs-datanode start
pssh –iH 'node1 node2 node3' service hadoop-hdfs-datanode status
```

Now let's check the size of our hdfs file system.

```
hdfs dfs -df –h
```

You should verify that the size of the file system is around 60 GB. Otherwise, something has gone wrong and you should check that your file systems are properly mounted on their datanodes.

The hadoop-hdfs-namenode-station*log file should show that each datanode successfully joined the cluster.

Step 5. Examine the HDFS cluster.

We can use commands implemented with hdfs dfsadmin to view the hdfs cluster.

```
su – hdfs
hdfs dfsadmin –printTopology
hdfs dfsadmin –report
```

```

```

These commands will give us a detailed look at the hdfs cluster.

Step 6. Install the secondary namenode.

First, we'll need to create a new directory to store the intermediate data files created by the Secondary Namenode when it does a checkpoint operation.

```
mkdir -m 700 -p /var/lib/hadoop-hdfs/namesecondary
chown hdfs:hadoop /var/lib/hadoop-hdfs/namesecondary
```

Now, on node 2, install the secondary namenode service.

```
yum install -y hadoop-hdfs-secondarynamenode
```

Start the service and verify that the initial checkpoint was performed.

```
service hadoop-hdfs-secondarynamenode start
sleep 60
grep -I fsimage /var/log/hadoop-hdfs/hadoop-hdfs-secondary-
namenode*.log
```

Now verify that the merged fs_image<oldnumber> was successfully transferred back to the namenode and that a new edit_inprogress file was created.

```
ls -l /var/lib/hadoop-hdfs/name/current
```

Run the following commands. The first one is needed to avoid a problem with a future exercise.

```
sed -I '/backup/d' /etc/fstab
umount /var/lib/hadoop-hdfs/backup/
```

Exercise 3. Uploading files in HDFS

Step 1. Upload files to HDFS.

Our first task is to create an upload directory for HDFS. Run the following commands.

```
su - hdfs
hdfs dfs -mkdir /upload
hdfs dfs -chmod 770 /upload
hdfs dfs -chgrp hdfsupload /upload
hdfs dfs -ls -d /upload
```

Additionally, we'll have to create an upload group in the /etc/group file.

```
groupadd hdfsupload
usermod -G hdfsupload <username>
id <username>
```

We'll use the local /etc directory as a sample data set to upload. Create a tar archive of this directory.

```
tar cf /tmp/etc.tar /etc/
ls -lh /tmp/etc.tar
```

Now we'll upload the tar archive to the hdfs directory.

```
su - <username>
id
hdfs dfs -put /tmp/etc.tar /upload/etc.tar
hdfs dfs -ls /upload/etc.tar
```

Note that the file is split into four blocks. Each block is replicated three times and spread between the data nodes.

Let's try uploading the file again, but this time forcing a larger blocksize of 64M.

```
hdfs dfs -D dfs.blocksize=64M -put /tmp/etc.tar
/upload/etc.tar.64Mblocksize
```

We can also change the replication factor to non-default values:

```
hdfs dfs –cp /upload/etc.tar /upload/etc/tar.2rep
hdfs dfs –setrep 2 /upload/etc.tar.2rep
hdfs dfs –cp /upload/etc.tar /upload/etc.tar.4rep
hdfs dfs –setrep 4 /upload/etc.tar.4rep
```

Note that running the hdfs dfs –ls option shows the replication factor in place of the link count as in a normal ls.

We can also use fsck to show us the integrity of the files and make sure that there is no file corruption present as well as show which nodes hold blocks for a given file.

```
su – hdfs
hdfs -fsck /upload
hdfs fsck /upload/etc.tar –files -blocks
```

We can emulate a disk failure by running a find and deleting specific blocks from the local datanode's storage using the following commands: We obtain the block number from the output of the hdfs fsck command.

```
find /var/lib/hadoop-hdfs/ -name 'blk_<block number>';
find /var/lib/hadoop-hdfs/ -name 'blk_<block number> -exec rm {} +;
```

However, if we re-run the hdfs fsck /upload/etc.tar file, it doesn't show the loss of the block as the hdfs fsck facility only examines the data within the namenode, not the data node.

Restart the datanode service on the data node where we deleted the block. Running a tail –f on the namenode log should show that it replicates the block again back to the datanode.

Exercise 4. Managing and maintaining HDFS

Step 1. Add datanodes to the namenode configuration.

Add the following properties to the /etc/hadoop/hdfs-site.xml file

Next, create a file that contains the IP address of your datanodes.

```
Touch /etc/hadoop/conf/dfs.hosts.exclude
(for i in 1 2 3: do ssh node$i hostname –i;  done) >
/etc/hadoop/conf/dfs.hosts
cat /etc/hadoop/conf/dfs.hosts
```

Now restart the namenode and verify that the datanodes have joined.

```
service hadoop-hdfs-namenode restart
su –c 'hdfs dfsadmin –report' hdfs | grep Live
```

If any of the datanodes are not listed as live, then recheck the contents of the dfs.hosts file and restart the service again.

Let's create a new, large, datafile for use in this exercise.

```
tar c /usr | su –c 'hdfs –put - /upload/node1.usr.tar ' <username>
```

Now let's verify that the data is currently spread out between the datanodes.

```
su –c 'hdfs dfsadmin –report' hdfs | egrep (Hostname|Used%) | tail –n
+2
```

We can also see what files are using what space on HDFS.

```
su – hdfs
hdfs fsck /upload/etc.tar.4rep –files –blocks –locations
hdfs fsck / –files –blocks –locations | grep –B1 <IP addr of your name
node>
```

Step 2. Use the HDFS balancer

We can also run the hdfs balancer. We'll start it with a threshold of two per cent.

```
hdfs balancer –threshold 2
```

Only run it for a few seconds before aborting it with a Control-C, otherwise it will take too long.

Step 3. Emulating disk failures and how to deal with them.

First, find a file with a block stored on the fourth data node.

```
su - hdfs
hdfs fsck /upload –files –blocks –locations | grep –B1 <IP address of
datanode” | grep ‘/’
exit
```

Now verify that the file is intact and readable.

```
su –c ‘hdfs dfs –cat /upload/etc.tar.4rep | tar tv’  <username>
```

Now, we’re going to change the block mapping for one of the logical volumes containing HDFS data so that it instead maps to a device that returns I/O errors. This will simulate a disk failure.

```
dmsetup table | grep hdfs[1234]
dmsetup wipe_table vg0-hdfs1
dmsetup table | grep hdfs[1234]
```

Note that the final command should show that vg0-hdfs1 now will map to the error device.

If we then try to reading a file from the first node like so:

```
su –c ‘hdfs dfs –get /upload/etc.tar.4rep /tmp’ <username>
```

This should work, because HDFS reads the blocks local to the HDFS client residing on the local data node.

Run the same command, this time from the fourth node.

It may or may not work, depending on whether the data has been cached in memory or must be fetched from the disk.

Flush the local caches and try reading again by running the following commands.

```
rm /tmp/etc.tar.4rep
sync && echo 3 > /proc/sys/vm/drop_caches
blockdev –flushbufs /dev/sda
su –c 'hdfs dfs –get /upload/etc.tar.4rep /tmp' <username>
ls –l /tmp/etc.tar.4rep
```

Note that we can see the error messages in the kernel log if we use the dmesg command.

```
dmesg | tail –n50 | grep error
```

Now let's restart the datanode service on the fourth node and look at the log files.

```
rm –rf /var/log/hadoop-hdfs/*log
service hadoop-hdfs-datanode restart
less +/WARN /var/log/hadoop-hdfs/hadoop-hdfs-datanode*log
```

Let's see if the namenode shows that the datanode is dead.

```
su –c •hdfs dfsadmin –report• hdfs | egrep •^Name|Last|Live|Dead•
```

Step 4. Tolerating disk failures on a datanode.

We can modify the Hadoop configuration to specify how many disks can fail and still allow the node to join the hadoop cluster.

We can specify this via a property in the /etc/hadoop/conf.4node/hdfs-site.xml file.

```
<property>
    <name>dfs.datanode.failed.volumes.tolerated</name>
    <value>2</value>
</property
```

The above setup tells Hadoop that it can tolerate up to two disk failures and still allow the node to join the cluster.

Let's push the new file out to all of the nodes in the cluster and restart the service.

```
pscp.pssh –H 'node2 node3 node4' /etc/hadoop-conf.4node/hdfs-site.xml /etc/hadoop/conf
```

```
allnodes 'service hadoop-hdfs-datanode restart'
su -c 'hdfs dfsadmin -report' hdfs | egrep 'Name|Last|Live|Dead'
```

Step 5. Decommissioning a datanode.

If a datanode has a disk failure, generally we want to decommission it as soon as possible.

We do this by running the following commands.

```
ssh node4 'hostname -I' >> /etc/hadoop/conf/dfs.hosts.exclude
su - hdfs
hdfs dfs admin -refreshNodes
hdfs dfsadmin -report | egrep 'Host|Deco'
sleep 60
hdfs dfs admin -report | egrep 'Host|Deco'
```

The final command generates a report showing the status of the datanodes decommissioning.

Note that it may take a bit of time for the host to be fully decommissioned.