In [**Hadoop**](http://data-flair.training/blogs/hadoop-introduction-tutorial-quick-guide/), HDFS(Hadoop distributed files system) is used for storing data. It has 2 components: Name node(master node) and Data node(Slave node). In Data node actual data is stored and name node stores the meta data that is the file location, block size, file permission. It also receives heart beats from live data nodes, so it is also responsible for sending a signal to replicate if data node is no more available.

**Metadata** is stored in memory of name node. So it is recommended to have large RAM for name node machines.

This Namenode is in safe mode

**Name node is in safe mode**. During start up, **Namenode** loads the filesystem state from fsimage and edits log file. ... During this time, **Namenode** stays in **safe mode**. A **Safemode** for **Namenode** is essentially a read-only **mode** for the HDFS cluster, where it does not allow any modifications to file system or blocks

**Safemode** in [**Apache Hadoop**](http://data-flair.training/blogs/hadoop-introduction-tutorial-quick-guide/) is a maintenance state of NameNode, during which NameNode doesn’t allow any modifications to the file system. In Safemode, [**HDFS**](http://data-flair.training/blogs/comprehensive-hdfs-guide-introduction-architecture-data-read-write-tutorial/) cluster is in read-only and doesn’t replicate or delete [**Data Blocks**](http://data-flair.training/blogs/data-blocks-hdfs-hadoop-distributed-file-system/).

**When NameNode starts:**

* It loads the file system namespace from the last saved ***FsImage*** into its main memory and the ***edits log*** file.
* Merges edits log file on fsimage and results in new file system namespace.
* Then it receives block reports containing information about block location from all datanodes. In SafeMode NameNode performs collection of block reports from datanodes. NameNode enters safemode automatically during its start up.  
  NameNode leaves Safemode after the DataNodes have reported that most blocks are available.

**To know the status of Safemode, use command:**  
hadoop dfsadmin –safemode get  
**To enter Safemode, use command:**  
bin/hadoop dfsadmin –safemode enter  
**To come out of Safemode, use command:**  
hadoop dfsadmin -safemode leave

**There are some prerequisite like you should have a working Hadoop multinode cluster (obviously you required a cluster because you are going to remove one or more Datanode whether it is temporary or permanent).**

We will start by adding Decommissioning property in Hadoop cluster. You need to add this for first time only and later on you need to update the exclude file alone. If the Decommissioning property is already added, then just update the exclude file for Decommissioning.

**Hadoop Cluster Configuration Note**– In my case Resource Manager and Namenode on different machine so run all command accordingly.

**Steps for Decommissioning:**

1) Before add any property, stop your cluster. Otherwise, it will affect your cluster. You can do this using the command *stop-dfs.sh*

stop-dfs.sh

Next, Go to your **Resource Manager** node to edit *yarn-site.xml*

2) You need to add this property in your **yarn-site.xml**

<property>

   <name>yarn.resourcemanager.node.exclude-path</name>

   <value>/home/hadoop/excludes</value>

</property>

Note- In value section, mention your excludes file address.

**Now, go to your master node (Namenode) and edit the** *hdfs-site.xml***file-**

3) Add this property to **hdfs-site.xml**

<property>

<name>dfs.hosts.exclude</name>

<value>/home/hadoop/excludes</value>

</property>

**Note**–*If the Resource Manager and the Namenode (Master Node) are on the same machine, then simply edit the yarn-site.xml  and hdfs-site.xml of Namenode (Master node)*

4) Next, start your cluster using the following commands:

 start-dfs.sh #(Run this command On Masternode/Namenode only)

 start-yarn.sh #(Run this command On Resource Manager)

**Note-***If the Resource Manager (Nodemanager) and Namenode are running on the same machine, then run the above commands on Namenode (Master Node) only.*

5) We need to update exclude file on **both machine** Resource manager and Namenode (Master Node), if it’s not there then we can create an exclude file on both the machines

vi excludes

Add the Datanode/Slave-node address, for decommissioning-

192.168.10.103

6) Run the following command in the Resource Manager:

yarn rmadmin -refreshNodes                     (on Resource Manager)

This command will basically check the yarn-site.xml and process that property.and decommission the mentioned node from yarn. It means now**yarn manager** will not give any job to this node.

7) Run the following command on the Namenode to check hdfs-site.xml and process the property and decommissioned the specified node/datanode.

hdfs dfsadmin -refreshNodes

hdfs dfsadmin -refreshNodes                 (on Namenode )

This command will basically check the *yarn-site.xml* and process that property, and Decommission the mentioned node from YARN. Meaning, the **YARN Manager** will not give any job to this node.

8) Run the command **hadoop dfsadmin –report**

 hadoop dfsadmin -report

**Commissioning of Datanodes:**

Commissioning process is just the opposite of decommissioning, but the configuration part is almost same for both.

**Follow the steps for commissioning configuration –**

Before starting commissioning steps, simply **remove the exclude file** on both machine or delete all the entries of exclude file ( make it blank)

**Stop all daemons before adding any property into Hadoop cluster.**

Open **Resource manager machine** to edit yarn-site.xml

1) Next, Go to yarn manager, and add this property into yarn-site.xml.

vi yarn-site.xml

<property>

   <name>yarn.resourcemanager.nodes.include-path</name>

<value>/home/hadoop/includes</value>

</property>

**Next, Go to your Namenode (Master Node).**

2) Add this property to hdfs-site.xml:

vi hdfs-site.xml (on Namenode )

<property>

 <name>dfs.hosts</name>

   <value>/home/hadoop/includes</value>

</property>

3) Now, start your cluster using the following commands:

 start-dfs.sh (Run this command On Namenode only)

start-yarn.sh (Run this command On Resource Manager)

**Note-***If the Resource Manager (Nodemanager) and the Namenode are running on same machine, then run these commands on Namenode (Master Node) only.*

4) We need to update the include file on both the Resource Manager and the Namenode (Master Node). If it’s not present, then create an include file on both the Nodes.

vi includes

Add the Datanode’s/Slave nodes IP address or hostname

192.168.10.101

192.168.10.102

192.168.10.103

**Note-***If you are going to add a new datanode or if you are scaling up your cluster by adding new node, you need to add the IP address and hostname to /etc/hosts file of all nodes ( Namenode, Datanode, Resource Manager).*

**Whenever you are going to do Commissioning, please mention all datanode address in the Include file.**

5) Run the following command on the Resource Manager

yarn rmadmin -refreshNodes         (on Resource Manager)

6) Next, go to the Master Node (Namenode) and run the following command to refresh all nodes:

Run this command to refresh all nodes-

hdfs dfsadmin -refreshNodes

7) Check Hadoop admin report using the command hadoop dfsadmin –report.

hadoop dfsadmin -report

**Here, you can see that dn3.mycluster.com (192.168.10.103) datanode, which was on decommissioned state, is now on the Normal state (Commissioned).**

***Note:***

* *The most important thing when you do commissioning is to make sure that the datanode which you are going to add has everything (Should be configured for Hadoop datanode).*
* *And second thing which you need to keep in your mind is that, you should have to mention all necessary datanodes address in the include files.*
* *Run cluster Balancer, as Balancer attempts to provide a balance to a certain threshold among data nodes by copying block data from older nodes to newly commissioned nodes.*

**How to run Hadoop Balancer?**

hadoop balancer

Hadoop Balancer is a built in property which makes sure that no datanode will be over utilized. When you run the balancer utility, it checks whether some datanode are under-utilized or over-utilized and will balance the replication factor. But make sure the Balancer should run in only off peak hours in a real cluster, because if you run this during peak hours, it will cause a heavy load to networking, as it will transfer large amount of data.

## **HDFS filesystem checking utility**

The command usage

$ hdfs fsck  
Usage: hdfs fsck <path> [-list-corruptfileblocks | [-move | -delete | -openforwrite] [-files [-blocks [-locations | -racks]]]] [-includeSnapshots] [-storagepolicies] [-blockId <blk\_Id>]  
    <path>    start checking from this path  
    -move    move corrupted files to /lost+found  
    -delete    delete corrupted files  
    -files    print out files being checked  
    -openforwrite    print out files opened for write  
    -includeSnapshots    include snapshot data if the given path indicates a snapshottable directory or there are snapshottable directories under it  
    -list-corruptfileblocks    print out list of missing blocks and files they belong to  
    -blocks    print out block report  
    -locations    print out locations for every block  
    -racks    print out network topology for data-node locations  
    -storagepolicies    print out storage policy summary for the blocks  
    -blockId    print out which file this blockId belongs to, locations (nodes, racks) of this block, and other diagnostics info (under replicated, corrupted or not, etc)

**Simple HDFS fsck**

$ hdfs fsck  /  
Connecting to namenode via http://<namenode>:50070/fsck?ugi=hadoop&path=%2F  
FSCK started by hadoop (auth:SIMPLE) from /192.168.0.1 for path / at Thu Dec 15 16:01:25 PST 2016  
.................................................................................................  
.................................................................................................  
..............................................................Status: HEALTHY  
 Total size:    238102023128 B  
 Total dirs:    17  
 Total files:    862  
 Total symlinks:        0  
 Total blocks (validated):    2261 (avg. block size 105308280 B)  
 Minimally replicated blocks:    2261 (100.0 %)  
 Over-replicated blocks:    0 (0.0 %)  
 Under-replicated blocks:    0 (0.0 %)  
 Mis-replicated blocks:        0 (0.0 %)  
 Default replication factor:    2  
 Average block replication:    1.985847  
 Corrupt blocks:        0  
 Missing replicas:        0 (0.0 %)  
 Number of data-nodes:        2  
 Number of racks:        1

## **How do I know if HDFS filesystem has corrupt blocks?**

The easiest way to determine this is to run an fsck on the filesystem. If you have setup your hadoop environment variables you should be able to use a path of /.

hdfs fsck /

or.

hdfs fsck hdfs://<namenode>:50070/

If the end of fsck output looks like this, you have corrupt blocks in your HDFS.

...  
/user/hadooptest/test59:  Under replicated BP-762523015-192.168.0.2-1480061879099:blk\_1073741976\_1152. Target Replicas is 2 but found 1 replica(s).  
  
/user/hadooptest/test59:  Under replicated BP-762523015-192.168.0.2-1480061879099:blk\_1073741977\_1153. Target Replicas is 2 but found 1 replica(s).  
.  
/user/hadooptest/test6: CORRUPT blockpool BP-762523015-192.168.0.2-1480061879099 block blk\_1073741845  
...  
 Total size:    238102023128 B  
 Total dirs:    17  
 Total files:    862  
 Total symlinks:        0  
 Total blocks (validated):    2261 (avg. block size 105308280 B)  
  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  UNDER MIN REPL'D BLOCKS:    32 (1.415303 %)  
  CORRUPT FILES:    11  
  MISSING BLOCKS:    32  
  MISSING SIZE:        3606911412 B  
  CORRUPT BLOCKS:     32  
  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 Corrupt blocks:        32  
 Number of data-nodes:        2  
 Number of racks:        1  
FSCK ended at Thu Dec 15 22:26:27 PST 2016 in 33 milliseconds

**Note**: In the example output above, you can see there are actually two types errors found

Under replicated  
CORRUPT blockpool

Let's try to fix them one by one

## **How to fix "Under replicated" files**

For this type of error, it's not a big issue. It's just means that a file that has 2 or more replicas, but now missing one copy of replica, to fix it, just try to make an extra copy for these files:

So, get the list of replicated files first

$ hdfs fsck / | grep -i "under replicated" | awk '{print $1}' |sort | uniq | sed -e 's/://g'> under\_replicated.flst

Then, run the command below to fix it

$ for f in `cat under\_replicated.flst` { echo "Fixing $f" ; hdfs dfs -setrep 3 $f; }

Example output:

...  
fixing /fibrevillage/usr/bin/catman  
Replication 3 set: /fibrevillage/usr/bin/catman  
fixing /fibrevillage/usr/bin/cc  
...

## **How do I know which files have corrupt blocks?**

The output of the fsck above will be very verbose, but it will mention which blocks are corrupt.

$ hdfs fsck / | egrep -v '^\.+$' | grep -i corrupt

Example output:

...  
/user/hadooptest/test9: CORRUPT blockpool BP-762523015-192.168.0.2-1480061879099 block blk\_1073741855  
/user/hadooptest/test9: CORRUPT blockpool BP-762523015-192.168.0.2-1480061879099 block blk\_1073741856  
/user/hadooptest/test9: MISSING 3 blocks of total size 339281920 B.Status: CORRUPT  
  CORRUPT FILES:    11  
  CORRUPT BLOCKS:     32  
 Corrupt blocks:        32  
The filesystem under path '/' is CORRUPT

Above is part of corrupted file list, and the output will not be a bunch of dots, and also files that might currently have under-replicated blocks (which isn't necessarily an issue). The output should include something like this with all your affected files.

## **How to locat corrupted block?**

The first step would be to gather information on the file's location, and blocks.

Find the corrupted file list:

$ hdfs fsck / | egrep -v '^\.+$' | grep -i "corrupt blockpool"| awk '{print $1}' |sort |uniq |sed -e 's/://g' >corrupted.flst

Run command below one by one(you could do it in a loop if situation is same)

$ hdfs fsck /user/hadooptest/test1 -locations -blocks -files  
FSCK started by hadoop (auth:SIMPLE) from /192.168.0.2 for path /user/hadooptest/test1 at Thu Dec 15 23:32:30 PST 2016  
/user/hadooptest/test1 339281920 bytes, 3 block(s):   
/user/hadooptest/test1: CORRUPT blockpool BP-762523015-192.168.0.2-1480061879099 block blk\_1073741830  
  
/user/hadooptest/test1: CORRUPT blockpool BP-762523015-192.168.0.2-1480061879099 block blk\_1073741831  
  
/user/hadooptest/test1: CORRUPT blockpool BP-762523015-192.168.0.2-1480061879099 block blk\_1073741832  
 MISSING 3 blocks of total size 339281920 B  
0. BP-762523015-192.168.0.2-1480061879099:blk\_1073741830\_1006 len=134217728 MISSING!  
1. BP-762523015-192.168.0.2-1480061879099:blk\_1073741831\_1007 len=134217728 MISSING!  
2. BP-762523015-192.168.0.2-1480061879099:blk\_1073741832\_1008 len=70846464 MISSING!

So, you can tell that the corrupted block is on the server 192.168.0.2

Log on the the node, check its datanode log, you will see something like this:

ExitCodeException exitCode=1: chmod: changing permissions of ‘/disk/c5t7’: Operation not permitted

This piece information tells you that the filesystem/block /disk/c5t7/ has i/o error

## **How to repair a corrupted file?**

From above section, we tracked down the node where the corruption is. Also, by looking through logs, we know where the problem was. If at this layer the problem can get fixed, and this piece of data directory can be brought back to online, then the error get fixed.

**Note:** this fix applies to "under replicas" error too

However, if underneath disk has media problem, can be replaced but data on it will get lost.The corrupted file more likely can't be recovered if there is no backup.

## **Remove the corrupted file from your hadoop cluster**

This command will move the corrupted file to the trash.

hdfs dfs -rm /path/to/corrupted.flst

Or you can skip the trash to permanently delete (which is probably what you want to do)

hdfs dfs -rm -skipTrash /path/to/corrupted.flst

**mapred.map.output.compression.codec**: I would use snappy

**mapred.output.compress**: This boolean flag will define is the whole map/reduce job will output compressed data. I would always set this to true also. Faster read/write speeds and less disk spaced used.

**mapred.output.compression.type**: I use block. This will make the compression splittable even for all compression formats (gzip, snappy, and bzip2) just make sure you're using a splitable file format like sequence, RCFile, or Avro.

**mapred.output.compression.codec**: this is the compression codec for the map/reduce job. I mostly use one of the three: Snappy (Fastest r/w 2x-3x compression), gzip (normal r fast w 5x-8x compression), bzip2 (slow r/w 8x-12x compression)