**Big data**

A large data that can be processed on simple database

**problems**

* storing (base of large volume)
* processing

data is divided and distributed among many individual databases

hdfs filesystem is a specially designed file system for storing huge dataset in commodity hardware it consist of name node and data node

**name node** - name node is the master daemon

there is only one active name node

manages the datanodes

stores all metadata (contains an mapping of where exactly the data is located) metadata consist of two files - editlog & fsimage

**edit log** - keeps a track of recent changes made on hdfs

only recent changes are tracked here

**fsimage** - keeps track of every change made on HDFS since

the beginning

(imp : when editlog file size increases namenode fails ,

sol: make copies of the editlog and fsimage file)

(we also create a secondary namenode)

secondary namenode - it is a node that maintains the copies of editlog and fsimage , it combines them both to get an updated version of the fsimage and push it to the primary name node .

**data node** - data node is the slave

there can be multiple data nodes

stores actual data

**ARCHITECTURE**

Rack switch - contains a rack of data nodes

core switch - it maintains the network bandwidth and connects the name node to the data node

\_rack switch \_datanode1

\_datanode1

........

\_ core switch \_rack switch \_datanode1

**NAMENODE.** \_datanode1

\_ core switch .........

\_rack switch \_datanode1

\_datanode1

..............

\_rack switch \_datanode1

\_datanode1

................

**data blocks:** hdfs splits massive files into small chunks , these chunks are known as data blocks .

* each file in hdfs is stored as a data block
* the default size of one data block is 128MB
* all the blocks are of same size except the last one (it can be same or lesser)

consider file having size 520 mb

**file =520 mb**

it will be stored as -> Block A= 128mb Block B=128mb

Block C=128mb Block D=128mb Block E=8mb

**Data node failure and replication** :

all the data blocks are stored in various DataNodes

The default replication factor is 3 i.e., in total, we will have 3 copies of each data block and shuffle it among data nodes

**Rack:** Rack is the collection of 30-40 datanodes . Rack Awareness is a concept that helps to decide where a replica of the data block should be stored .

**heart beat:** it is the signal that DataNodes continuously send to the Namenode .this signal shows the status of the data node operation

* the client first request the NameNode to read the data
* the namenode allows the client to read the requested data from the DataNodes and sent to client
* update the metadata
* block ops : performs block creation, deletion and replication

**hadoop file system read mechanism**

* client opens up a connection with Distributed file system
* then gets the block location from name node by using remote procedure call
* the namenode checks if the client is authorized to access the requested file (if yes, it then provides the block location & a token to the client which is shown to the slave for authentication) .
* the client will show the authentication token to the Data Node for the read process to begin (hdfs client read through fsdata input steam )
* after reaching the data block the connection is closed

**hadoop file system write mechanism**

* the client first executes create file on the distributed file system
* the dfs interacts with the namenode to create a file . namenode then provides the location to write the data
* it writes the data through fsdata output stream
* the data is written and the slave further replicates it
* ack is given to the namenode if it is successful and then client performs close method

—----------------------------------------------------------------------------------------------------------------------------

## **Prerequisites**

* *VIRTUAL BOX*: it is used for installing the operating system on it.
* *OPERATING SYSTEM*: You can install Hadoop on Linux-based operating systems. Ubuntu and CentOS are very commonly used. In this tutorial, we are using CentOS.
* *JAVA*: You need to install the Java 8 package on your system.
* *HADOOP*: You require the Hadoop 2.7.3 package.

## **Install Hadoop**

### Step 1: [Click here](https://goo.gl/ipdJJa) to download the **Java 8 Package**. Save this file in your home directory.

### 

### Step 2: Extract the Java Tar File.

*Command*: **tar -xvf jdk-8u101-linux-i586.tar.gz**

### Step 3: Download the Hadoop 2.7.3 Package.

*Command*:

**wget** [**https://archive.apache.org/dist/hadoop/core/hadoop-2.7.3/hadoop-2.7.3.tar.gz**](https://archive.apache.org/dist/hadoop/core/hadoop-2.7.3/hadoop-2.7.3.tar.gz)

### Step 4: Extract the Hadoop tar File.

*Command*: **tar -xvf hadoop-2.7.3.tar.gz**

### Step 5: Add the Hadoop and Java paths in the bash file (.bashrc).

Open. bashrc file. Now, add Hadoop and Java Path as shown below.

Learn more about the Hadoop Ecosystem and its tools with the [Hadoop Certification](https://www.edureka.co/big-data-hadoop-training-certification).

*Command*:  **vi .bashrc**



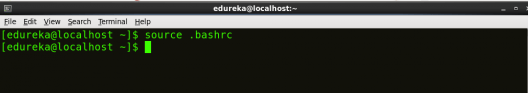
**export JAVA\_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.352.b08-2.el7\_9.x86\_64/jre**

**Eg:-** export JAVA\_HOME=/usr/local/Cellar/openjdk@8/1.8.0+345/libexec/openjdk.jdk/Contents/Home/jre

Then, save the bash file and close it.

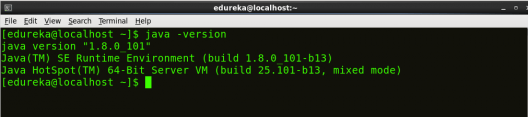
For applying all these changes to the current Terminal, execute the source command.

*Command*: **source .bashrc**



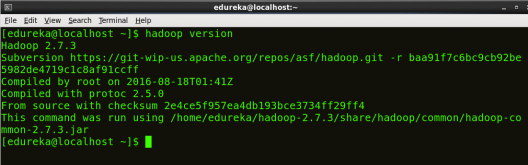
To make sure that Java and Hadoop have been properly installed on your system and can be accessed through the Terminal, execute the java -version and hadoop version commands.

*Command*: **java -version**



*Fig: Hadoop Installation – Checking Java Version*

*Command*: **hadoop version**

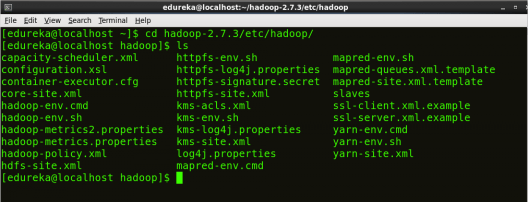


### Step 6: Edit the [Hadoop Configuration files](https://www.edureka.co/blog/explaining-hadoop-configuration/).

*Command:* **cd hadoop-2.7.3/etc/hadoop/**

*Command:* **ls**

All the Hadoop configuration files are located in hadoop-2.7.3/etc/hadoop directory as you can see

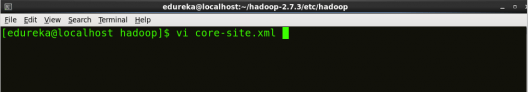


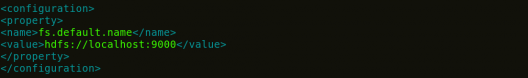
*Fig: Hadoop Installation – Hadoop Configuration Files*

### Step 7: Open *core-site.xml* and edit the property mentioned below inside configuration tag:

*core-site.xml* informs Hadoop daemon where NameNode runs in the cluster. It contains configuration settings of Hadoop core such as I/O settings that are common to HDFS & MapReduce.

*Command*: **vi core-site.xml**





*Fig: Hadoop Installation – Configuring core-site.xml*

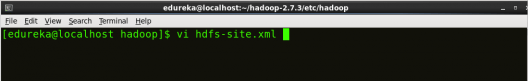
| *<?xml version="1.0" encoding="UTF-8"?>*  *<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>*  *<configuration>*  *<property>*  *<name>fs.default.name</name>*  *<value>hdfs://localhost:9000</value>*  *</property>*  *</configuration>* |
| --- |

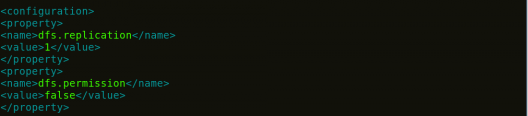
### 

### Step 8: Edit *hdfs-site.xml* and edit the property mentioned below inside configuration tag:

*hdfs-site.xml* contains configuration settings of HDFS daemons (i.e. NameNode, DataNode, Secondary NameNode). It also includes the replication factor and block size of HDFS.

*Command*: **vi hdfs-site.xml**





*Fig: Hadoop Installation – Configuring hdfs-site.xml*

| <?xml version="1.0" encoding="UTF-8"?>  <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>  <configuration>  <property>  <name>dfs.replication</name>  <value>1</value>  </property>  <property>  <name>dfs.permission</name>  <value>false</value>  </property>  </configuration> |
| --- |

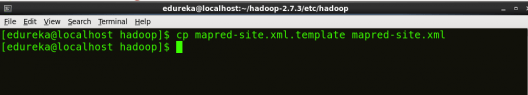
### Step 9: Edit the *mapred-site.xml* file and edit the property mentioned below inside configuration tag:

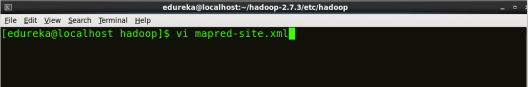
*mapred-site.xml* contains configuration settings of MapReduce applications like number of JVM that can run in parallel, the size of the mapper and the reducer process, CPU cores available for a process, etc.

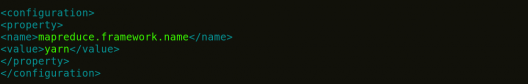
In some cases, mapred-site.xml file is not available. So, we have to create the mapred-site.xml file using the mapred-site.xml template.

*Command*: **cp mapred-site.xml.template mapred-site.xml**

*Command*: **vi mapred-site.xml.**







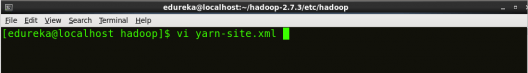
*Fig: Hadoop Installation – Configuring mapred-site.xml*

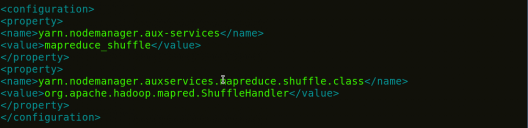
| <?xml version="1.0" encoding="UTF-8"?>  <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>  <configuration>  <property>  <name>mapreduce.framework.name</name>  <value>yarn</value>  </property>  </configuration> |
| --- |

### Step 10: Edit *yarn-site.xml* and edit the property mentioned below inside configuration tag:

*yarn-site.xml* contains configuration settings of ResourceManager and NodeManager like application memory management size, the operation needed on program & algorithm, etc.

*Command*: **vi yarn-site.xml**





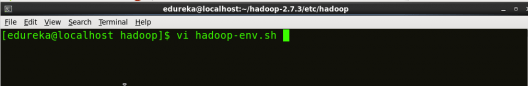
*Fig: Hadoop Installation – Configuring yarn-site.xml*

| <?xml version="1.0">  <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> |
| --- |

### Step 11: Edit *hadoop-env.sh* and add the Java Path as mentioned below:

*hadoop-env.sh* contains the environment variables that are used in the script to run Hadoop like Java\_home path, etc.

*Command*: **vi hadoop–env.sh**



Property of hadoop-env - Install Hadoop - Edureka

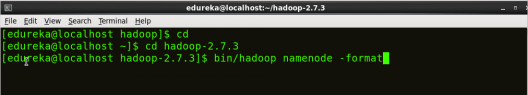
*Fig: Hadoop Installation – Configuring hadoop-env.sh*

### Step 12: Go to the Hadoop home directory and format the NameNode.

*Command*: **cd**

*Command*: **cd hadoop-2.7.3**

*Command*: **bin/hadoop namenode -format**



*Fig: Hadoop Installation – Formatting NameNode*

This formats the HDFS via NameNode. This command is only executed for the first time. Formatting the file system means initializing the directory specified by the dfs.name.dir variable.

Never format, up and running Hadoop filesystem. You will lose all your data stored in the HDFS.

### Step 13: Once the NameNode is formatted, go to hadoop-2.7.3/sbin directory and start all the daemons.

*Command:* **cd hadoop-2.7.3/sbin**

Either you can start all daemons with a single command or do it individually.

*Command:* ***./*start-all.sh**

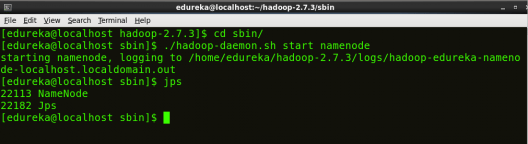
The above command is a combination of***start-dfs.sh, start-yarn.sh* & *mr-jobhistory-daemon.sh***

Or you can run all the services individually as below:

### **Start NameNode:**

The NameNode is the centerpiece of an HDFS file system. It keeps the directory tree of all files stored in the HDFS and tracks all the files stored across the cluster.

*Command:* **./hadoop-daemon.sh start namenode**

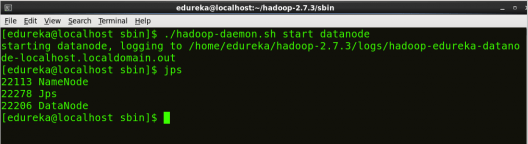


*Fig: Hadoop Installation – Starting NameNode*

### **Start DataNode:**

On startup, a DataNode connects to the Namenode and it responds to the requests from the Namenode for different operations.

*Command:***./hadoop-daemon.sh start datanode**

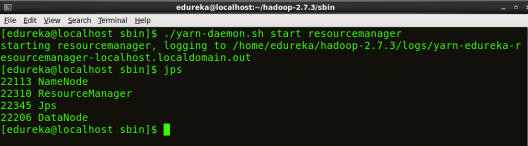


*Fig: Hadoop Installation – Starting DataNode*

### **Start ResourceManager:**

ResourceManager is the master that arbitrates all the available cluster resources and thus helps in managing the distributed applications running on the YARN system. Its work is to manage each NodeManagers and each application’s ApplicationMaster.

*Command:* **./yarn-daemon.sh start resourcemanager**

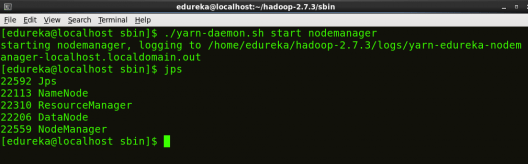


*Fig: Hadoop Installation – Starting ResourceManager*

### **Start NodeManager:**

The NodeManager in each machine framework is the agent which is responsible for managing containers, monitoring their resource usage and reporting the same to the ResourceManager.

*Command:* **./yarn-daemon.sh start nodemanager**



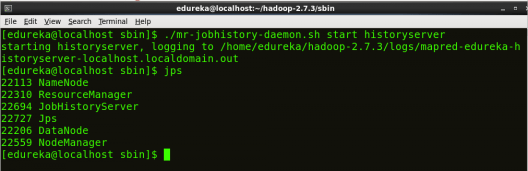
### **Start JobHistoryServer:**

JobHistoryServer is responsible for servicing all job history related requests from clients.

*Command*: **./mr-jobhistory-daemon.sh start historyserver**

### Step 14: To check that all the Hadoop services are up and running, run the below command.

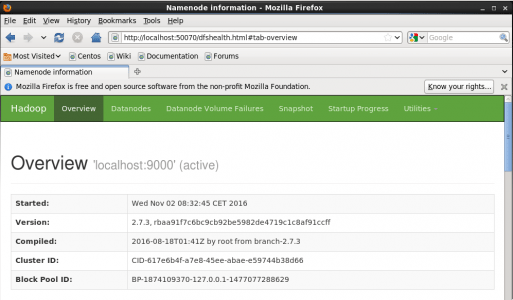
*Command:***jps**



*Fig: Hadoop Installation – Checking Daemons*

### Step 15: Now open the Mozilla browser and go to

### **localhost:50070/dfshealth.html** to check the NameNode interface.



*Fig: Hadoop Installation – Starting WebUI*

Congratulations, you have successfully installed a **single-node Hadoop cluster** in one go.

—----------------------------------------------------------------------------------------------------------------------------

**hadoop fs/hdfs dfs COMMANDS (CLI) :-**

* **hdfs dfs -ls** (to list all hadoop files)
* **hdfs dfs -put filename** (to put a file from our machine to hdfs)
* **hdfs dfs -put file dir\_name** (put a file from local to hdfs dir dir\_name)
* **hadoop fs -moveFromLocal abc.txt dir\_1** (to put a file from our machine to hdfs)
* **hdfs dfs -cat pw** (to see the path of file system)
* **hdfs dfs -cat full-path-tofile** (to view the file)
* **hdfs dfs -mv file file.new** (to rename the file)
* **hdfs dfs -rm filename** (to remove a file)
* **hdfs dfs -mkdir dir\_name**(to make a dir)
* **hdfs dfs --help** (provides all the commands or operations)
* **hadoop fs -touchz abc.txt** (to create a empty file in home folder of hdfs)
* **hadoop fs -copyFromLocal abc.txt dir\_name** (to copy file from local to hdfs in dir)
* **hadoop fs -copyToLocal dir/abc.txt ~/Desktop/** (to copy file from hdfs to local in dir)
* **hadoop fs -cp dir/abc.txt dir\_1** (to copy files within two hdfs directories)
* **hadoop fs -cp file:///home/ghalyan/dir/abc.txt dir\_1**
* **hadoop fs -mv dir\_1/abc.txt dir\_2** (to move files within two hdfs directories)
* **hadoop fs -du** (to check which files/dir. Is taking how much space){**du -s**}
* **hadoop fs -getmerge -nl dir\_1/file\_1.txt dir\_2/file\_2.txt ~/Desktop/mergedfile.txt**

(here we first mentioned both files then at last where we want the merged file to be saved ,

merged file will be created automatically , all three files are in hdfs server itself , besides that **-getmerge** is to merge and **-nl** is to add new line)

* **hadoop fs -appendToFile file\_1.txt file\_2.txt file\_3.txt**

(here we just appended data of two files into third which is from local to hdfs)

* **hadoop fs -checksum dir\_1/abc.txt**

(-checksum used to verify the integrity of file , can know file is modified or not using hash value)

* **hadoop fsck - /** (to check health of filesystem/any location)
* **hadoop fs -count dir\_1** (to check total no. of files in dir and size of directory)
* **hadoop fs -chgrp root dir\_1/abc.txt** (changed group of file)
* **hadoop fs -head dir\_1/abc.txt** (shows first few lines of file)
* **hadoop fs -expunge** (delete all the files in trash)
* **hadoop fs -chown aman:aman dir\_1/abc.txt** (changed both group and owner of file)
* **hadoop fs -setrep -w 3 dir\_2/abc.txt**

(setting up the replication factor){-w is wait till the replication is completed}

* **hadoop fs -truncate -w 100 /var/log/messages**

(it’s used to reduce the size of the file )

* **echo $?** (it checks the below cmd output is true/false){**0 - true, 1 - false**}
* **hadoop fs -test -d dir\_1** (check whether it’s a directory or not) eg:- **file or dir.**
* **hadoop fs -test -e dir\_2** (check whether mentioned file/dir. exists or not)
* **hadoop fs -test -z dir\_1/abc.txt** (check file is empty/non-empty)

**PREFIX:-**

* **hadoop fs**

It can of hadoop file system , cloud fs, local fs or any other fs

* **hdfs dfs**

This is specifically about hadoop distributed file system

* **XML**

Extensible Markup Language

[Hadoop Configuration Files](https://youtu.be/WHfSiJQmkac)

[Hadoop XML files Configuration](https://youtu.be/LQdD5l447Ks)

[HDFS - Important Configuration Parameters](https://youtu.be/rCqQGnQdXSQ)

[Hadoop Multi Node Cluster Setup](https://youtu.be/_iP2Em-5Abw)

[Hadoop Single Node Setup](https://youtu.be/98UCknD8_qA)

[Hadoop File Read Operation | How to read data in HDFS | Hadoop HDFS tutorial](https://youtu.be/2E-rjuKVEh8)

[HDFS Tutorial | Hadoop File Write Operation | How to write data in HDFS](https://youtu.be/hwCh4OMjLK0)

* Where datanode is storing the data Change it to /data/1
* Configure the capacity to 50 gb
* Test some HDFS commands to add and remove files,directories
* Learn about HDFS commands to ownership, users in hdfs

**Journal nodes**

The Role of **Journal nodes** is to keep both the Namenodes in sync and avoid hdfs split brain scenario by allowing only Active NN to write into journals.

In order for the Standby node to keep its state coordinated with the Active node, both nodes communicate with a group of separate daemons called ‘JournalNodes’ (JNs). When any namespace modification is performed by the Active node, it logs a record of the changes made, in the JournalNodes. The Standby node is capable of reading the amended information from the JNs, and is regularly monitoring them for changes. As the Standby Node sees the changes, it then applies them to its own namespace. In case of a failover, the Standby will make sure that it has read all the changes from the JounalNodes before changing its state to ‘Active state’. This guarantees that the namespace state is fully synched before a failover occurs.

**JournalNode machines** - the machines on which you run the JournalNodes. The JournalNode daemon is relatively lightweight, so these daemons may reasonably be collocated on machines with other Hadoop daemons, for example NameNodes, the JobTracker, or the YARN ResourceManager. ***Note: There must be at least 3 JournalNode daemons, since edit log modifications must be written to a majority of JNs. This will allow the system to tolerate the failure of a single machine***. You may also run more than 3 JournalNodes, but in order to actually increase the number of failures the system can tolerate, you should run an odd number of JNs, (i.e. 3, 5, 7, etc.). Note that when running with N JournalNodes, the system can tolerate at most (N - 1) / 2 failures and continue to function normally.

“Fencing” avoids such scenarios. Fencing is a process of ensuring that only one NameNode remains active at a particular time.

## Hadoop\_NameNode\_High\_Availability\_Architecture

## How to achieve High Availability in Hadoop?

The solution to protect yourself from any failure is the backup. That's it. In this case, we need to make a backup of two things.

1. HDFS namespace information.  
   All the information that a name node maintains should be continuously backed up at some other place. So that in the case of a failure, you have all the necessary information to start a new name node on a different machine.
2. Standby Name node machine.  
   To minimize the time to start a new name node, we should already have a standby computer pre configured and ready to take over the role of name node.

Now, let's come to the namespace information backup. We already learned that the name node maintains the entire file system in memory and we call it in memory *fsImage*. Name node also maintains an edit log in its local disk. Every time the name node makes a change in the filesystem. It records that change in the *editLog*. The *editLog* is like a journal ledger of name node. If we have the *editLog*, we can reconstruct the in-memory *fsImage*. So, we need to make a backup of the name node *editLog*.

**But the question is where and how?**

## Hadoop Quorum Journal Manager

## Standby Hadoop Name Node

## Hadoop Zookeeper failover controller

**zookeeper failover controller**

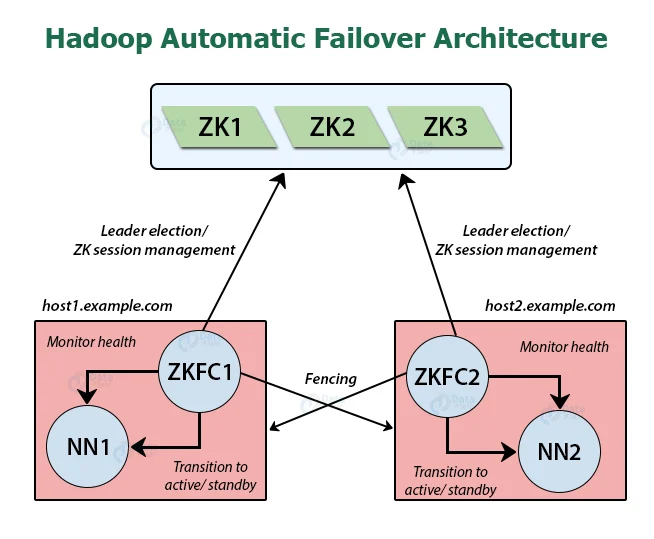
**There are two types of failover, that is, Graceful Failover and Automatic Failover.**

#### 1. Graceful Failover

#### 2. Automatic Failover

The two new components are deployed to Hadoop HDFS for implementing Automatic Failover. These two components are-

1. ZooKeeper quorum
2. ZKFailoverController process(ZKFC)



### 1. ZooKeeper quorum

#### a. Failure detection

#### b. Active NameNode election

### 2. ZKFailoverController(ZKFC)

1. **Health monitoring**
2. **ZooKeeper session management**
3. **ZooKeeper-based election**

### Configuring automatic failover

When the cluster is running, it is not possible to transfer from a manual failover setup to an automatic failover setup. So, before configuring automatic failover, the [Hadoop](https://hadoop.apache.org/) cluster should be shut down.

For configuring automatic failover, we need to add two new parameters.

1. In hdfs-site.xml file, add:

<property>

<name>dfs.ha.automatic-failover.enabled</name>

<value>**true**</value>

</property>

This parameter specifies that the cluster is now set for automatic failover.

2. In core-site.xml file, add:

<property>

<name>ha.zookeeper.quorum</name> <value>zk1.example.com:2181,zk2.example.com:2181,zk3.example.com:2181</value>

</property>

This parameter lists the host-port pairs which are running the ZooKeeper service.

After adding the configuration keys, we have to initialize required state in ZooKeeper using below command from one of the NameNode hosts:

[hdfs]$ $HADOOP\_HOME/bin/zkfc -formatZK

It will create a lock “znode” in the ZooKeeper inside which the automatic failover system stores its data.

Now, start the cluster with start-dfs.sh that will automatically start a ZKFC daemon on any machine that runs a NameNode.

ZKFCs now automatically selects one of the NameNodes to become active.

**high availability(epoch no.)**

When a NameNode becomes active, it is assigned an integer epoch number. Each epoch number is unique. No two NameNode have the same epoch number. When a NameNode sends any message to a JournalNode, it includes its epoch number as part of the request.