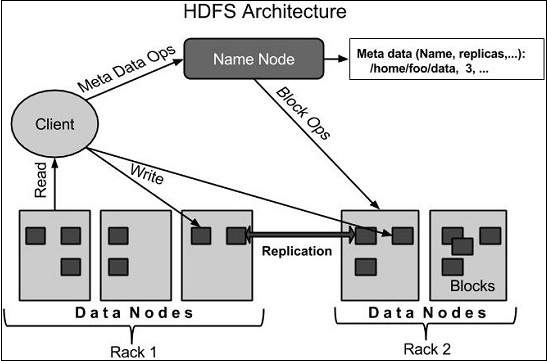
**What is HDFS**

Hadoop comes with a distributed file system called HDFS. In HDFS data is distributed over several machines and replicated to ensure their durability to failure and high availability to parallel application.

It is cost effective as it uses commodity hardware. It involves the concept of blocks, data nodes and node name.

**HADOOP Architecture**

Given below is the architecture of a Hadoop File System.



HDFS follows the master-slave architecture and it has the following elements.

### **Namenode**

The namenode is the commodity hardware that contains the GNU/Linux operating system and the namenode software. It is a software that can be run on commodity hardware. The system having the namenode acts as the master server and it does the following tasks −

* Manages the file system namespace.
* Regulates client’s access to files.
* It also executes file system operations such as renaming, closing, and opening files and directories.

### **Datanode**

The datanode is a commodity hardware having the GNU/Linux operating system and datanode software. For every node (Commodity hardware/System) in a cluster, there will be a datanode. These nodes manage the data storage of their system.

Datanodes perform read-write operations on the file systems, as per client request.

They also perform operations such as block creation, deletion, and replication according to the instructions of the namenode.

### **Block**

Generally, the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes. These file segments are called as blocks. In other words, the minimum amount of data that HDFS can read or write is called a Block. The default block size is 64MB, but it can be increased as per the need to change in HDFS configuration.

**Namenode Importance**

For any file system,especially HDFS, it is important to store its metadata reliably. while the file data is accessed in a write once and read many model, the metadata structures (e.g., the names of files and directories) can be modified by a large number of clients concurrently. It is important that this information is never desynchronized. Therefore, it is all handled by a single machine, called the **NameNode**.

The NameNode stores all the metadata for the file system. Because of the relatively low amount of metadata per file (it only tracks file names, permissions, and the locations of each block of each file), all of this information can be stored in the main memory of the NameNode machine, allowing fast access to the metadata.

The below two files are stored in Name Node local file system.

**Fsimage:**The persistent storage of file system file (metadata store – namespace) called as Fsimage which contains mapping of blocks to files and its properties which stores in name node local file system.

**EditLog:** The transaction log which also persistent record every change occurs in the file system called as Editlog file which also stored in Name Node local file system.

**Datanode Responsibilities**

DataNodes store data in a Hadoop cluster and is the name of the daemon that manages the data. File data is replicated on multiple DataNodes for reliability and so that localized computation can be executed near the data.

Within a cluster, DataNodes should be uniform. If they are not uniform, issues can occur. For example, DataNodes with less memory fill up more quickly than DataNodes with more memory, which can result in job failures.

**Important**

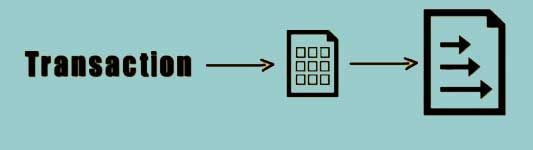
The default replication factor for HDFS is three. That is, three copies of data are maintained at all times. Cloudera recommends that you do not configure a lower replication factor when you have at least three DataNodes. A lower replication factor may lead to data loss.

* [**How NameNode manages blocks on a failed DataNode**](https://docs.cloudera.com/runtime/7.2.10/hdfs-overview/topics/hdfs-how-namenode-manages-blocks-on-a-failed-datanode.html)  
  A DataNode is considered dead after a set period without any heartbeats (10.5 minutes by default).
* [**Replace a disk on a DataNode host**](https://docs.cloudera.com/runtime/7.2.10/hdfs-overview/topics/hdfs-replacing-a-disk-on-a-datanode-host.html)  
  In your CDP Private Cloud Base cluster, you can replace faulty disks on the DataNode host. You must stop all the managed services and decommission the DataNode role instance before replacing the faulty disk.
* [**Remove a DataNode**](https://docs.cloudera.com/runtime/7.2.10/hdfs-overview/topics/hdfs-removing-a-datanode.html)  
  Before removing a DataNode, ensure that all the prerequisites for removal are satisfied.
* [**Fixing block inconsistencies**](https://docs.cloudera.com/runtime/7.2.10/hdfs-overview/topics/hdfs-fixing-block-inconsistencies.html)  
  You can use the output of hdfs fsck or hdfs dfsadmin -report commands for information about inconsistencies with the HDFS data blocks such as missing, misreplicated, or underreplicated blocks. You can adopt different methods to address these inconsistencies.
* [**Add storage directories using Cloudera Manager**](https://docs.cloudera.com/runtime/7.2.10/hdfs-overview/topics/hdfs-adding-storage-directories-using-cloudera-manager.html)  
  You can add a new storage directory and specify the storage type using Cloudera Manager.
* [**Remove storage directories using Cloudera Manager**](https://docs.cloudera.com/runtime/7.2.10/hdfs-overview/topics/hdfs-removing-storage-directories-using-cloudera-manager.html)  
  You can use Cloudera Manager to remove existing storage directories and specify new directories.
* [**Configuring storage balancing for DataNodes**](https://docs.cloudera.com/runtime/7.2.10/hdfs-overview/topics/hdfs-configuring-storage-balancing-for-datanodes.html)  
  You can configure HDFS to distribute writes on each DataNode in a manner that balances out available storage among that DataNode's disk volumes.
* [**Perform a disk hot swap for DataNodes using Cloudera Manager**](https://docs.cloudera.com/runtime/7.2.10/hdfs-overview/topics/hdfs-performing-disk-hot-swap-for-datanodes-using-cloudera-manager.html)  
  You can replace disks on your CDP Private Cloud Base cluster without shutting down a DataNode. This is referred to as hot swap.

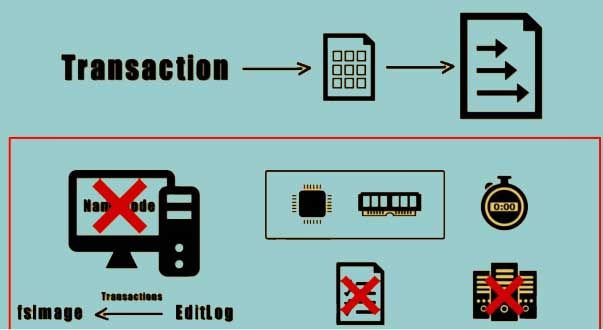
**Secondary Namenode**

Secondary namenode **keeps the checkpoint on the namenode, It reads the edit logs from the namenode continuously after a specific interval and applies it to the fsimage copy of secondary namenode**. In this way the fsimage file will have the most recent state of HDFS.

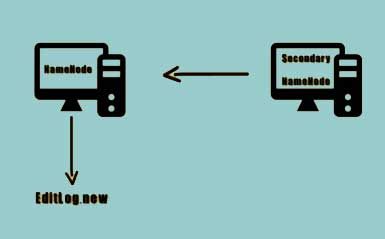
Each and every transaction that occurs on the file system is recorded within the edit log file. At some point of time this file becomes very large.



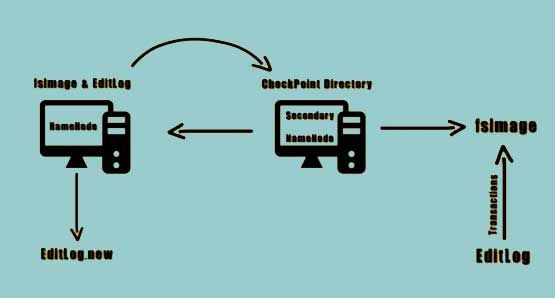
In such cases, if the Namenode fails due to corrupted meta data or any other reason than It has to retrieve the FS image from the disk and apply all the transactions to it present within the edit log file. In order to apply all these transactions, the system resources should be available. It also takes lot of time to apply all these transactions. Until all these transactions are not applied the contents of the FS image are inconsistent hence the cluster they cannot be operational.



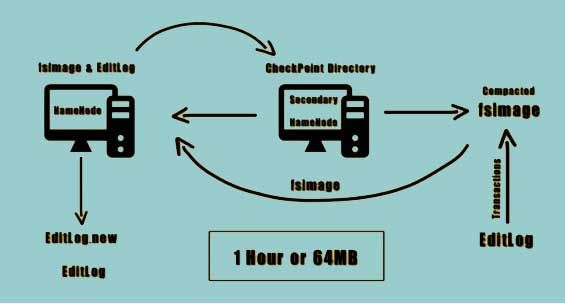
The secondary Namenode can be used to prevent the situation from occurring. The secondary Namenode instructs the Namenode to record the transactions to a new edit log file



The secondary Namenode copies the FS image and the edit log file to its checkpoint directory. Once these files are copied the secondary Namenode loads they FS image and applies all the transactions from the edit log file and stores this information onto a new and compacted FS image file.



The secondary Namenode transfers this compacted FS image file to the Namenode. The Namenode adopts this new FS image file and also renames the new edit log file that was created back to edit log file. This process occurs every hour or whenever the size of the added Logfile reaches to 64 MB.

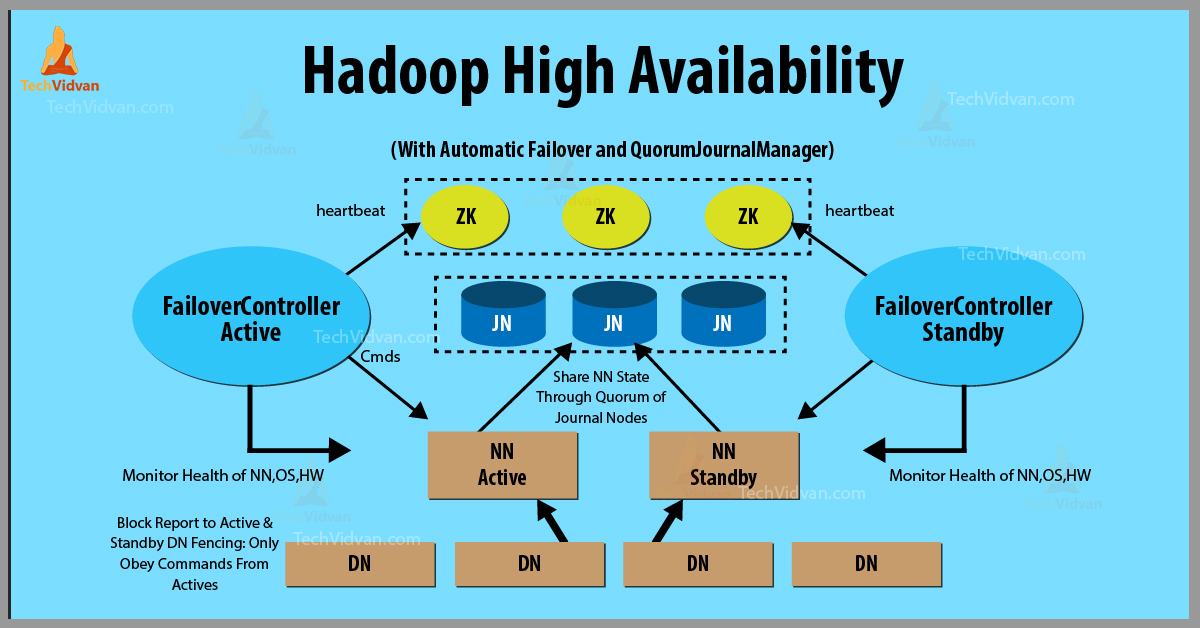


**High Availability**

**Hadoop HDFS** is a distributed file system. HDFS distributes data among the nodes in the Hadoop cluster by creating a replica of the file. Hadoop framework store these replicas of files on the other machines present in the cluster.

So, when an HDFS client wants to access his data, he can easily access that data from a number of machines present in the cluster. Data is easily available in the closest node in the cluster.

At some unfavorable conditions like a failure of a node, the client can easily access their data from the other nodes. This feature of Hadoop is called**High Availability**



volume is gedempt

### **How is High Availability achieved in Hadoop?**

In the HDFS cluster, there is a number of **DataNodes**. After the definite interval of time, all these DataNodes sends heartbeat messages to the **NameNode**. If the NameNode stops receiving heartbeat messages from any of these DataNodes, then it assumes it to be dead.

After that, it checks for the data present in those nodes and then gives commands to the other datanode to create a replica of that data to other datanodes. Therefore data is always available.

When a client asks for a data access in HDFS, first of all, NameNode searches for the data in that datanodes, in which data is quickly available. And then provides access to that data to the client.

Clients don’t have to search for the data in all the datanodes. HDFS Namenode itself makes data availability easy to the clients by providing the address of the datanode from where a user can directly read.

### **Example of Hadoop High Availability**

Hadoop HDFS provides High availability of data. When the client requests NameNode for data access, then the NameNode searches for all the nodes in which that data is available.

After that, it provides access to that data to the user from the node in which data was quickly available.While searching for data on all the nodes in the cluster, if NameNode finds some node to be dead, then without user knowledge NameNode redirects the user to the other node in which the same data is available.

Without any interruption, data is made available to the user. So in conditions of node failure also data is highly available to the users.