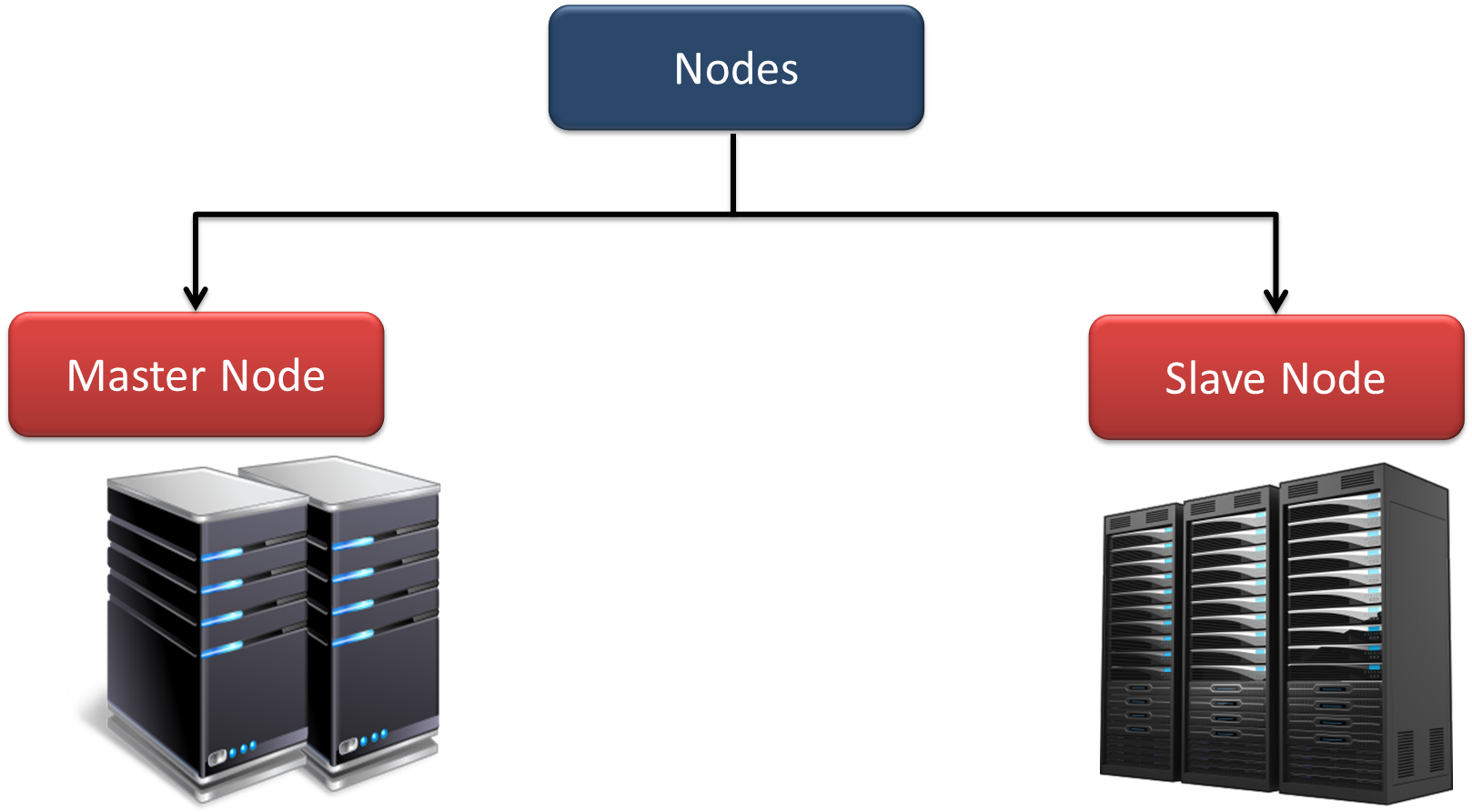
**HDFS Tutorial – Introduction**

**Hadoop Distributed File system – HDFS** is the world’s most reliable storage system. HDFS is a Filesystem of Hadoop designed for storing very large files running on a cluster of commodity hardware. It is designed on principle of storage of less number of large files rather than the huge number of small files. Hadoop HDFS also provides fault tolerant storage layer for Hadoop and its other components. HDFS Replication of data helps us to attain this feature. It stores data reliably even in the case of hardware failure. It provides high throughput access to application data by providing the data access in parallel. Let us move ahed in this Hadoop HDFS tutorial with major areas of Hadoop Distributed File System.

## HDFS Nodes

As we know Hadoop works in**master-slave** fashion, HDFS also has 2 types of nodes that work in the same manner. There are **namenode(s)** and **datanodes** in the cluster.

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/HDFS-Nodes-tutorial.png)

*Hadoop HDFS Tutorial – Hadoop HDFS Nodes*

### HDFS Master (Namenode)

Namenode regulates file access to the clients. It maintains and manages the slave nodes and assign tasks to them. Namenode executes file system namespace operations like opening, closing, and renaming files and directories. It should be deployed on reliable hardware.

### HDFS Slave (Datanode)

There are a number of slaves or datanodes in Hadoop Distributed File System which manage storage of data. These slave nodes are the actual worker nodes which do the tasks and serve read and write requests from the file system’s clients. They also perform block creation, deletion, and replication upon instruction from the NameNode. Once a block is written on a datanode, it replicates it to other datanode and process continues until the number of replicas mentioned is created. Datanodes can be deployed on commodity Hardware and we need not deploy them on very reliable hardware.

## Hadoop HDFS Daemons

There are 2 daemons which run for HDFS for data storage:

* **Namenode:** This is the daemon that runs on all the masters. Name node stores metadata like filename, the number of blocks, number of replicas, a location of blocks, block IDs etc. This metadata is available in memory in the master for faster retrieval of data. In the local disk, a copy of metadata is available for persistence. So name node memory should be high as per the requirement.
* **Datanode:** This is the daemon that runs on the slave. These are actual worker nodes that store the data.

## Data storage in HDFS

Whenever any file has to be written in HDFS, it is broken into small pieces of data known as blocks. HDFS has a default block size of **128 MB** which can be increased as per the requirements. These blocks are stored in the cluster in distributed manner on different nodes. This provides a mechanism for **MapReduce**to process the data in parallel in the cluster. To learn more about How data flows in Hadoop MapReduce, **follow this MapReduce tutorial**.

*Hadoop HDFS – Data Storage*

Multiple copies of each block are stored across the cluster on different nodes. This is a replication of data. By default, HDFS replication factor is 3. It provides fault tolerance, reliability, and **high availability**.

A Large file is split into n number of small blocks. These blocks are stored at different nodes in the cluster in a distributed manner. Each block is replicated and stored across different nodes in the cluster.

## Rack Awareness in Hadoop HDFS

Hadoop runs on a cluster of computers which are commonly spread across many racks. NameNode places replicas of a block on multiple racks for improved fault tolerance. NameNode tries to place at least one replica of a block in each rack, so that if a complete rack goes down then also system will be highly available Optimizing replica placement distinguishes HDFS from most other distributed file systems. The purpose of a rack-aware replica placement policy is to improve data reliability, availability, and network bandwidth utilization.

## Hadoop HDFS Feature

In the HDFS tutorial, let us now see various features of Hadoop Distributed File System

### a. Distributed Storage

As HDFS stores data in a distributed manner. It divides the data into small pieces and stores it in different nodes of the cluster. In this manner, Hadoop Distributed File System provides a way to map reduce to process a subset of large data, which is broken into smaller pieces and stored in multiple nodes, parallelly on several nodes. MapReduce is the heart of Hadoop but HDFS is the one which provides it all these capabilities.

### b. Blocks

As HDFS splits huge files into small chunks known as blocks. Block is the smallest unit of data in a filesystem. We (client and admin) do not have any control on the block like block location. Namenode decides all such things.

HDFS default block size is 128 MB which can be increased as per the requirement. This is unlike OS filesystem where the block size is 4 KB.

### c. Replication

Hadoop HDFS creates duplicate copies of each block. This is called replication. All blocks are replicated and stored at different nodes across the cluster. It tries to put at least 1 replica in every rack.

What do you mean by rack?

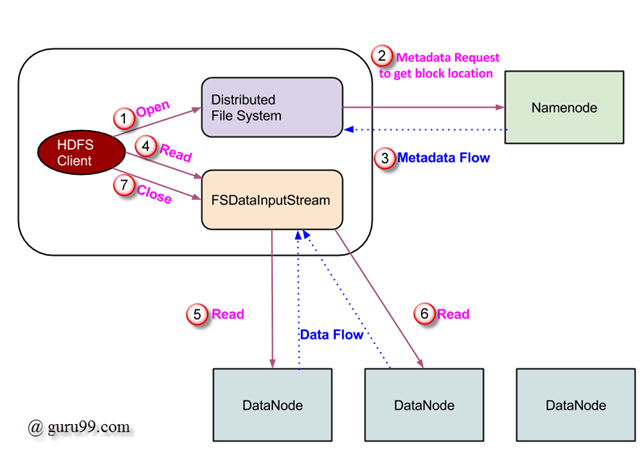
Datanodes are arranged in racks. All the nodes in a rack are connected by a single switch so if a switch or complete rack is down, data can be accessed from another rack. We will see it further in rack awareness section.

As seen earlier in this Hadoop HDFS tutorial, default replication factor is 3 and this can be changed to the required values according to the requirement by editing the configuration files (hdfs-site.xml)

HDFS Operations

**Read Operation In HDFS**

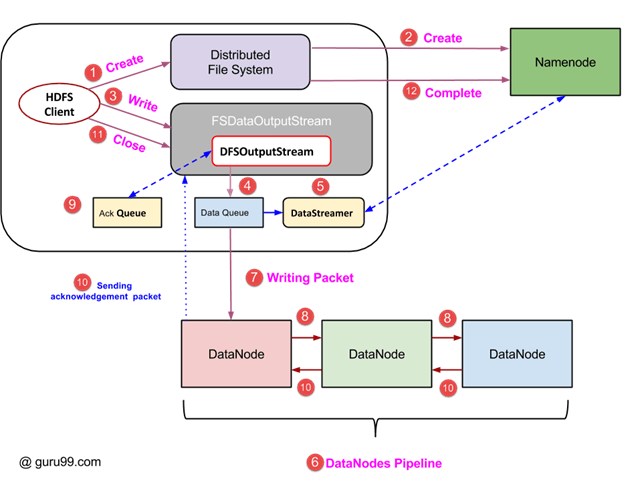
Data read request is served by HDFS, NameNode and DataNode. Let's call reader as a 'client'. Below diagram depicts file read operation in Hadoop.

[](https://cdn.guru99.com/images/Big_Data/061114_0923_LearnHDFSAB1.png)

1. Client initiates read request by calling **'open()'** method of FileSystem object; it is an object of type **DistributedFileSystem**.
2. This object connects to namenode using RPC and gets metadata information such as the locations of the blocks of the file. Please note that these addresses are of first few block of file.
3. In response to this metadata request, addresses of the DataNodes having copy of that block, is returned back.
4. Once addresses of DataNodes are received, an object of type **FSDataInputStream** is returned to the client. **FSDataInputStream** contains **DFSInputStream** which takes care of interactions with DataNode and NameNode. In step 4 shown in above diagram, client invokes **'read()'** method which causes **DFSInputStream** to establish a connection with the first DataNode with the first block of file.
5. Data is read in the form of streams wherein client invokes **'read()'** method repeatedly. This process of **read()** operation continues till it reaches end of block.
6. Once end of block is reached, DFSInputStream closes the connection and moves on to locate the next DataNode for the next block
7. Once client has done with the reading, it calls **close()** method.

**Write Operation In HDFS**

In this section, we will understand how data is written into HDFS through files.

[](https://cdn.guru99.com/images/Big_Data/061114_0923_LearnHDFSAB2.png)

1. Client initiates write operation by calling 'create()' method of DistributedFileSystem object which creates a new file - Step no. 1 in above diagram.
2. DistributedFileSystem object connects to the NameNode using RPC call and initiates new file creation. However, this file create operation does not associate any blocks with the file. It is the responsibility of NameNode to verify that the file (which is being created) does not exist already and client has correct permissions to create new file. If file already exists or client does not have sufficient permission to create a new file, then **IOException** is thrown to client. Otherwise, operation succeeds and a new record for the file is created by the NameNode.
3. Once new record in NameNode is created, an object of type FSDataOutputStream is returned to the client. Client uses it to write data into the HDFS. Data write method is invoked (step 3 in diagram).
4. FSDataOutputStream contains DFSOutputStream object which looks after communication with DataNodes and NameNode. While client continues writing data, **DFSOutputStream** continues creating packets with this data. These packets are en-queued into a queue which is called as **DataQueue**.
5. There is one more component called **DataStreamer** which consumes this **DataQueue**. DataStreamer also asks NameNode for allocation of new blocks thereby picking desirable DataNodes to be used for replication.
6. Now, the process of replication starts by creating a pipeline using DataNodes. In our case, we have chosen replication level of 3 and hence there are 3 DataNodes in the pipeline.
7. The DataStreamer pours packets into the first DataNode in the pipeline.
8. Every DataNode in a pipeline stores packet received by it and forwards the same to the second DataNode in pipeline.
9. Another queue, 'Ack Queue' is maintained by DFSOutputStream to store packets which are waiting for acknowledgement from DataNodes.
10. Once acknowledgement for a packet in queue is received from all DataNodes in the pipeline, it is removed from the 'Ack Queue'. In the event of any DataNode failure, packets from this queue are used to reinitiate the operation.
11. After client is done with the writing data, it calls close() method (Step 9 in the diagram) Call to close(), results into flushing remaining data packets to the pipeline followed by waiting for acknowledgement.
12. Once final acknowledgement is received, NameNode is contacted to tell it that the file write operation is complete.

start-dfs.sh

hdfs dfs -ls /

hadoop fs -mkdir <paths>

hadoop fs -put /home/saurzcode/Samplefile.txt /user/saurzcode/dir3/

hadoop fs -get /user/saurzcode/dir3/Samplefile.txt /home/

hadoop fs -cat /user/saurzcode/dir1/abc.txt

hadoop fs -cp /user/saurzcode/dir1/abc.txt /user/saurzcode/dir2

hadoop fs -copyFromLocal /home/saurzcode/abc.txt /user/saurzcode/abc.txt

hadoop fs -mv /user/saurzcode/dir1/abc.txt /user/saurzcode/dir2

hadoop fs -rm /user/saurzcode/dir1/abc.txt

hadoop fs -rmr /user/saurzcode/

hadoop fs -tail /user/saurzcode/dir1/abc.txt

hadoop fs -du /user/saurzcode/dir1/abc.txt

hdfs dfs -touchz /test2/file1.txt

hdfs dfs -getmerge -nl /test1 file1.txt

hdfs dfs -chmod -R 777 /new-dir

hdfs dfs -chown -R admin:hadoop /new-dir

hadoop fs -mkdir <paths>

