UNIT-II

DISTRIBUTED FILE SYSTEMS LEADING TO HADOOP FILE SYSTEM

Big Data :

'Big Data' is also a **data** but with a **huge size**.

'Big Data' is a term used to describe collection of data that is huge in size and yet growing exponentially with time.

Data which are very large in size is called Big Data.

Normally we work on data of size MB(Word Doc, Excel) or maximum GB(Movies, Codes) but data in Peta bytes i.e. 10^15 byte size is called Big Data.

It is stated that almost 90% of today's data has been generated in the past 3 years.

In short, such a data is so large and complex that none of the traditional data management tools are able to store it or process it efficiently.

Examples Of 'Big Data'

Following are some the examples of 'Big Data'-

The **New York Stock Exchange** generates about ***one terabyte*** of new trade data per day.

Social Media Impact

Statistic shows that ***500+terabytes*** of new data gets ingested into the databases of social media site **Facebook,**Google, LinkedIn, every day. This data is mainly generated in terms of photo and video uploads, message exchanges, putting comments etc.

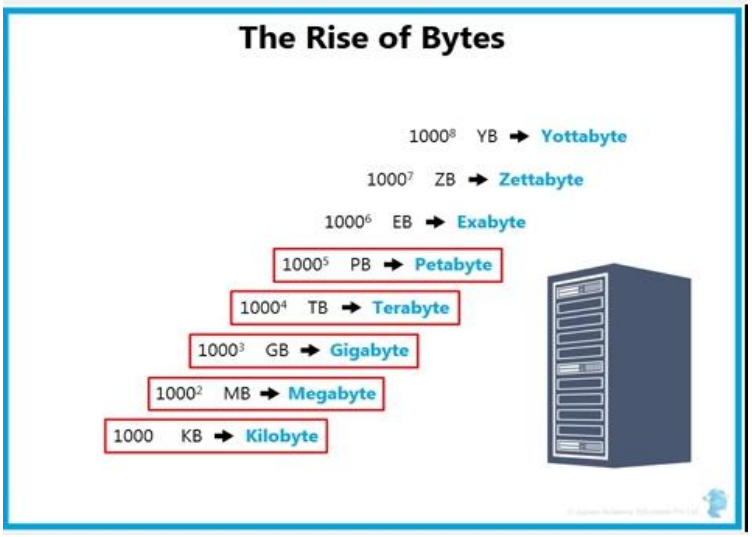
Single **Jet engine** can generate ***10+terabytes*** of data in ***30 minutes*** of a flight time. With

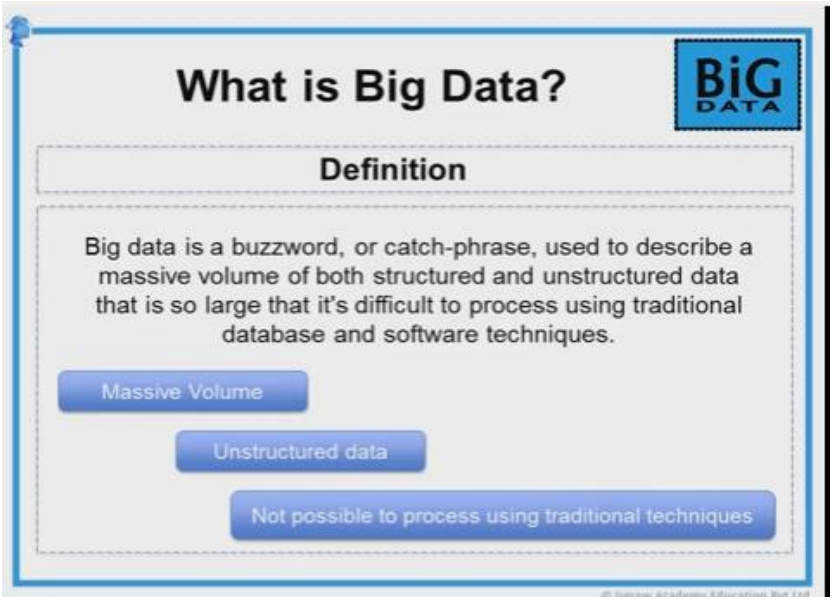
many thousand flights per day, generation of data reaches up to many ***Petabytes***.

E-commerce site: Sites like Amazon, Flipkart, Alibaba generates huge amount of logs from which users buying trends can be traced.

Weather Station: All the weather station and satellite gives very huge data which are stored and manipulated to forecast weather.

Telecom company: Telecom giants like Airtel, Vodafone study the user trends and accordingly publish their plans and for this they store the data of its million users.





Categories Of 'Big Data'

Big data' could be found in three forms:

1. **Structured**
2. **Unstructured**
3. **Semi-structured**

Structured

Any data that can be stored, accessed and processed in the form of fixed format is termed as a 'structured' data.

Do you know? 10^{21} bytes equals to ***1 zettabyte*** or ***one billion terabytes*** forms ***a zettabyte***.

Data stored in a relational database management system is one example of a **'structured'** data.

Examples Of Structured Data

An 'Employee' table in a database is an example of Structured Data

Employee_ID	Employee_Name	Gender	Department	Salary_In_lacs
2365	Rajesh Kulkarni	Male	Finance	650000
3398	Pratibha Joshi	Female	Admin	650000
7465	Shushil Roy	Male	Admin	500000
7500	Shubhojit Das	Male	Finance	500000
7699	Priya Sane	Female	Finance	550000

Unstructured

Any data with unknown form or the structure is classified as unstructured data.

Typical example of unstructured data is, a heterogeneous data source containing a combination of simple text files, images, videos etc.

Examples Of Un-structured Data

Output returned by 'Google Search' .

Semi-structured

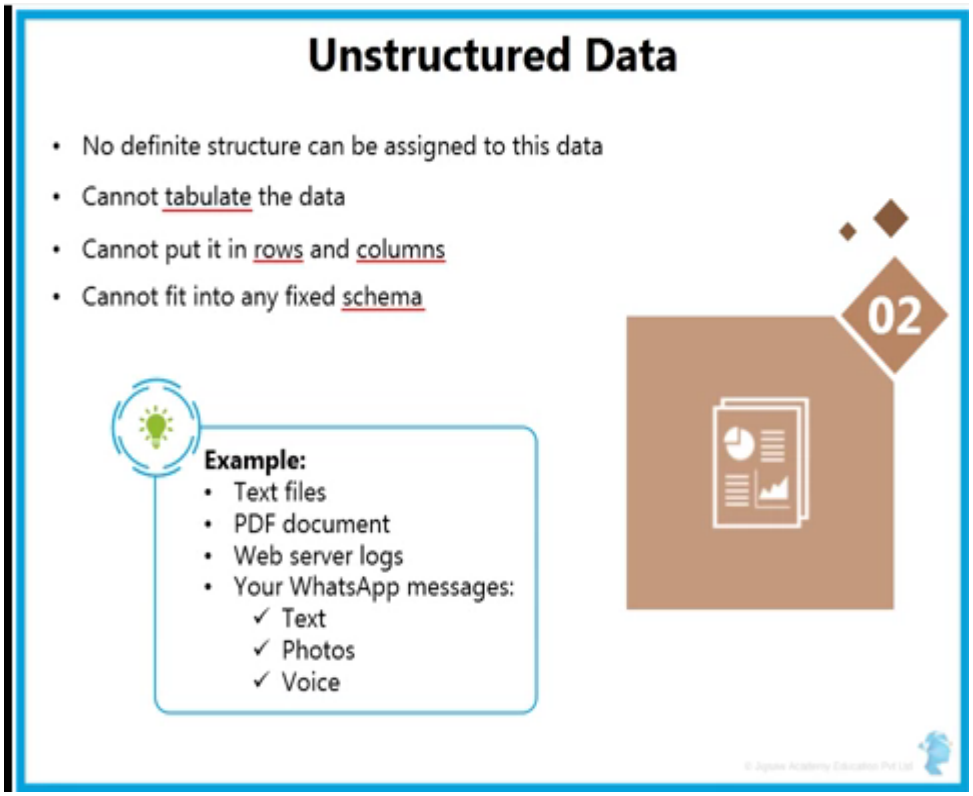
Semi-structured data can contain both the forms of data.

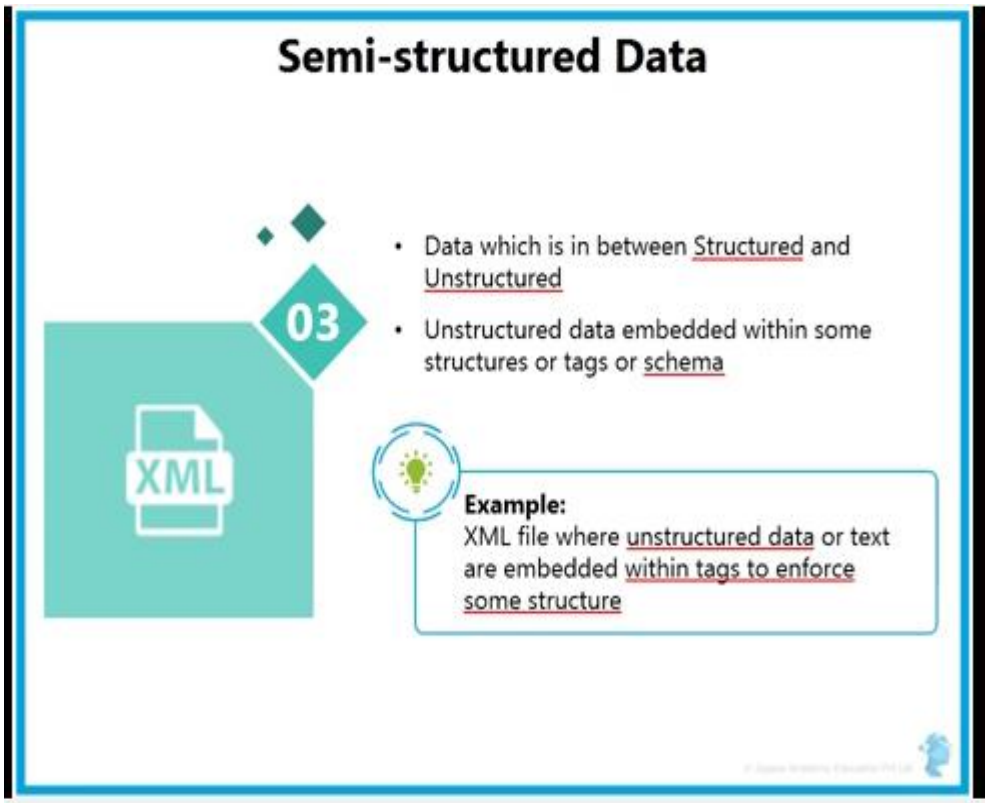
Example of semi-structured data is a data represented in XML file.

Examples Of Semi-structured Data

Personal data stored in a XML file-

<rec><name>Prashant Rao</name><sex>Male</sex><age>35</age></rec>
<rec><name>Seema R.</name><sex>Female</sex><age>41</age></rec>
<rec><name>Satish Mane</name><sex>Male</sex><age>29</age></rec>
<rec><name>Subrato Roy</name><sex>Male</sex><age>26</age></rec>
<rec><name>Jeremiah J.</name><sex>Male</sex><age>35</age></rec>


Unstructured Data
No definite structure can be assigned to this data
Cannot tabulate the data
Cannot put it in rows and columns
Cannot fit into any fixed schema
02
Example:
Text files
PDF document
Web server logs
Your WhatsApp messages:
Text
Photos
Voice


Semi-structured Data
03
XML
Data which is in between Structured and Unstructured
Unstructured data embedded within some structures or tags or schema
Example:
XML file where unstructured data or text are embedded within tags to enforce some structure

Characteristics of big data:

Volume

Variety

Velocity

Volume' is one characteristic which needs to be considered while dealing with 'Big Data'.

Size of data plays very crucial role in determining value out of data.

Variety

Variety refers to heterogeneous sources and the nature of data, both structured and unstructured.

During earlier days, spreadsheets and databases were the only sources of data considered by most of the applications.

Now days, data in the form of emails, photos, videos, monitoring devices, PDFs, audio, etc. is also being considered in the analysis applications.

Velocity

The term **'velocity'** refers to the speed of generation of data.

Big Data Velocity deals with the speed at which data flows in from sources like business processes, application logs, networks and social media sites, sensors, Mobile devices, etc. The flow of data is massive and continuous.

Benefits of Big Data Processing

Ability to process 'Big Data' brings in multiple benefits, such as-

- **Businesses can utilize outside intelligence while taking decisions**.

- **Improved customer service.**

• **Early identification of risk to the product/services, if any**.

• **Better operational efficiency**.

Issues

Huge amount of unstructured data which needs to be stored, processed and analyzed.

Solution:

Apache Hadoop is the most important framework for working with Big Data. The biggest strength of Hadoop is scalability.

Background of Hadoop

- With an increase in the penetration of internet and the usage of the internet, the data captured by Google increased exponentially year on year.
- Just to give you an estimate of this number, in 2007 Google collected on an average 270 PB of data every month.
- The same number increased to 20000 PB everyday in 2009.
- Obviously, Google needed a better platform to process such an enormous data.
- Google implemented a programming model called MapReduce, which could process this 20000 PB per day. Google ran these MapReduce operations on a special file system called Google File System (GFS). Sadly, GFS is not an open source.

Doug cutting and Yahoo! reverse engineered the model GFS and built a parallel Hadoop Distributed File System (HDFS).

The software or framework that supports HDFS and MapReduce is known as Hadoop.

What is Hadoop

Hadoop is an open source framework from Apache software foundation.

Hadoop is written in Java and is not OLAP (online analytical processing).

It is used for batch/offline processing.

It is used to store process and analyze data which are very huge in volume.

It is being used by Facebook, Yahoo, Google, Twitter, LinkedIn and many more.

Why to use Hadoop

Apache Hadoop is not only a storage system but is a platform for data storage as well as processing.

It is **scalable** (as we can add more nodes on the fly).

Fault tolerant (Even if nodes go down, data processed by another node).

It efficiently processes large volumes of data on a cluster of commodity hardware.

Hadoop is for processing of huge volume of data.

Commodity hardware is the low-end hardware, they are cheap devices which are very economical. Hence, Hadoop is very economic.

The idea of Apache Hadoop was actually born out of a Google project called the MapReduce, which is a framework for breaking down an application into smaller chunks that can then be parsed on a much smaller and granular level. Each of the smaller blocks is individually operated on nodes which are then connected to the main cluster.

Hadoop works in **master-slave** fashion. There is a master node and there are n numbers of slave nodes where n can be 1000s.

Master manages, maintains and monitors the slaves.

slaves are the actual worker nodes. Master should deploy on good configuration hardware, not just commodity hardware. As it is the centerpiece of **Hadoop cluster**.

Master stores the metadata (data about data) while slaves are the nodes which store the data.

Distributedly data stores in the cluster.

The client connects with master node to perform any task.

Hadoop components

1)Hdfs -storage

HDFS :, Hadoop uses HDFS (Hadoop Distributed File System) which uses commodity hardware to form clusters and store data in a distributed fashion.

It works on Write once, read many times principle.

2) Map_Reduce-processing.

Map Reduce paradigm is applied to data distributed over network to find the required output.

HDFS architecture:

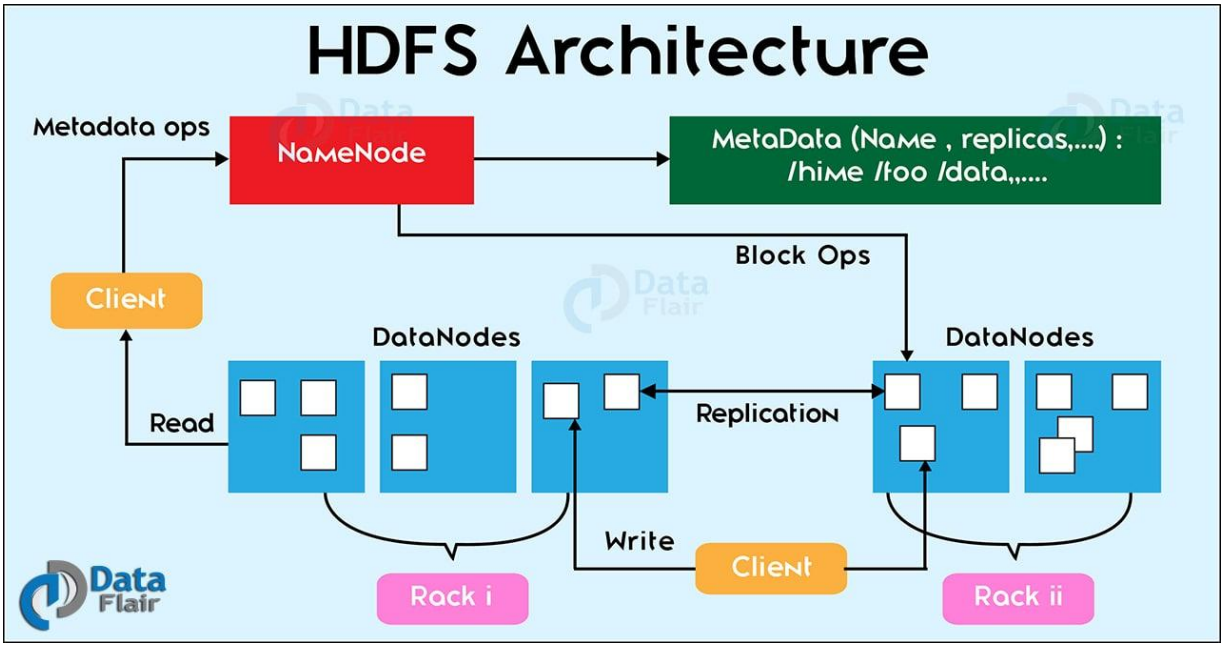
Hadoop comes with a distributed file system called **HDFS** (**HADOOP Distributed File Systems**) HADOOP based applications make use of HDFS.

HDFS is designed for storing very large data files, running on clusters of commodity hardware.

It is fault tolerant, scalable, and extremely simple to expand.

Hadoop HDFS has a **Master/Slave** architecture in which *Master* is **NameNode** and *Slave* is **DataNode**.

HDFS Architecture consists of single NameNode and all the other nodes are DataNodes.



HDFS NameNode

It is also known as *Master node*.

HDFS Namenode stores *meta-data* i.e. number of **data blocks**, replicas and other details.

This meta-data is available in memory in the master for faster retrieval of data. NameNode maintains and manages the slave nodes, and assigns tasks to them. It should deploy on reliable hardware as it is the centerpiece of HDFS.

Task of NameNode

- Manage file system namespace.
- Regulates client's access to files.
- It also executes file system execution such as naming, closing, opening files/directories.
- All DataNodes sends a Heartbeat and block report to the NameNode in the **Hadoop cluster.**
- It ensures that the DataNodes are alive. A block report contains a list of all blocks on a datanode.
- NameNode is also responsible for taking care of the *Replication Factor* of all the blocks.

Files present in the NameNode metadata are as follows-

FsImage –

It is an "Image file". **FsImage** contains the entire filesystem namespace and stored as a file in the namenode's local file system.

It also contains a serialized form of all the directories and file inodes in the filesystem.

Each *inode* is an internal representation of file or directory's metadata.

EditLogs –

It contains all the recent modifications made to the file system on the most recent FsImage.

Namenode receives a create/update/delete request from the client.

After that this request is first recorded to edits file.

HDFS DataNode

It is also known as *Slave*.

In Hadoop HDFS **Architecture**, DataNode stores actual data in HDFS.

It performs read and write operation as per the request of the client.

DataNodes can deploy on commodity hardware.

Task of DataNode

- Block replica creation, deletion, and replication according to the instruction of Namenode.
- DataNode manages data storage of the system.
- DataNodes send heartbeat to the NameNode to report the health of HDFS. By default, this frequency is set to 3 seconds.

Secondary Namenode: Secondary NameNode downloads the FsImage and EditLogs from the NameNode.

And then merges EditLogs with the FsImage (FileSystem Image).

It keeps edits log size within a limit.

It stores the modified FsImage into persistent storage.

And we can use it in the case of NameNode failure.

Secondary NameNode performs a regular checkpoint in HDFS.

Checkpoint Node

The **Checkpoint node** is a node which periodically creates checkpoints of the namespace.

Checkpoint Node in Hadoop first downloads FsImage and edits from the Active Namenode.

Then it merges them (FsImage and edits) locally, and at last, it uploads the new image back to the active NameNode.

Backup Node

In Hadoop, Backup node keeps an in-memory, up-to-date copy of the file system namespace.

The Backup node checkpoint process is more efficient as it only needs to save the namespace into the local FsImage file and reset edits.

NameNode supports one Backup node at a time.

Blocks

HDFS in **Apache Hadoop** split huge files into small chunks known as *Blocks.* These are 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.

The default size of the HDFS block is 64 MB, which we can configure as per the need.

All blocks of the file are of the same size except the last block, which can be the same size or smaller.

The major advantages of storing data in such block size are that it saves disk seek time.

Replication Management

Block replication provides **fault tolerance**. If one copy is not accessible and corrupted then we can read data from other copy.

The number of copies or replicas of each block of a file is **replication factor**. The default replication factor is 3 which are again configurable. So, each block replicates three times and stored on different DataNodes.

If we are storing a file of 128 MB in HDFS using the default configuration, we will end up occupying a space of 384 MB (3*128 MB).

NameNode receives block report from DataNode periodically to maintain the replication factor.

When a block is over-replicated/under-replicated the NameNode add or delete replicas as needed.

Rack Awareness

In a large **cluster of Hadoop**, in order to improve the network traffic while reading/writing HDFS file, NameNode chooses the DataNode which is closer to the same rack or nearby rack to Read /write request. NameNode achieves rack information by maintaining the rack ids of each DataNode. **Rack Awareness** in Hadoop is the concept that chooses Datanodes based on the rack information.

In **HDFS Architecture**, NameNode makes sure that all the replicas are not stored on the same rack or single rack. It follows *Rack Awareness Algorithm* to reduce latency as well as fault tolerance. We know that default replication factor is 3.

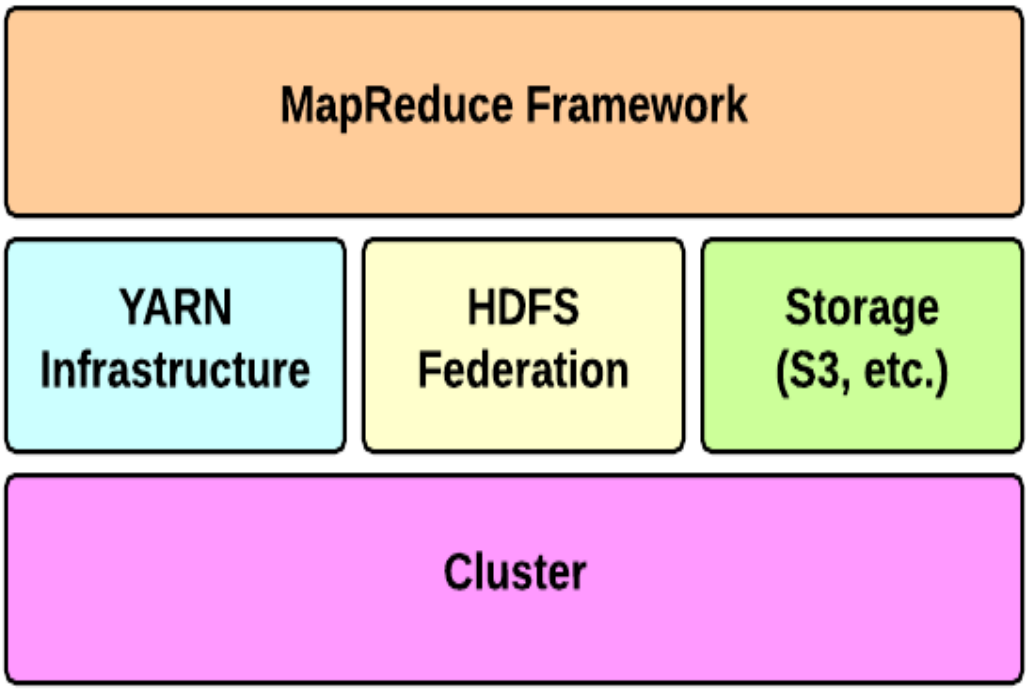
According to *Rack Awareness Algorithm* first replica of a block will store on a local rack. The next replica will store another datanode within the same rack. The third replica will store on different rack In Hadoop.

Rack Awareness is important to improve:

- Data **high availability** and reliability.
- The performance of the cluster.
- To improve network bandwidth.

Hadoop Architecture Overview

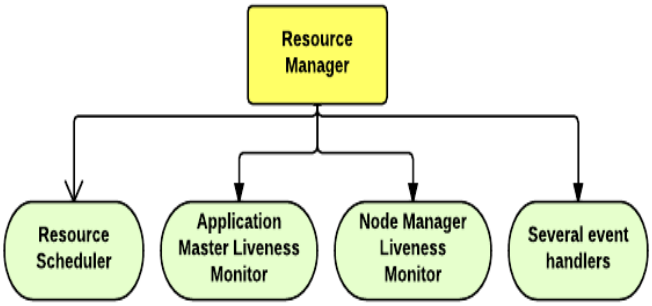
Apache Hadoop is an open-source software framework for storage and large-scale processing of data-sets on clusters of commodity hardware. There are mainly five building blocks inside this runtime envinroment (from bottom to top):



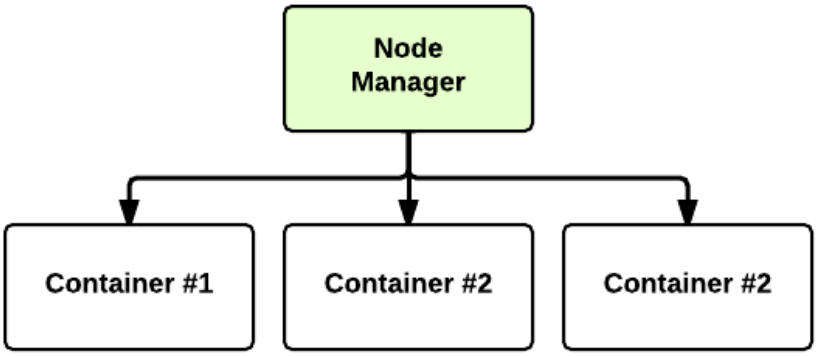
- The cluster is the set of host machines (nodes). Nodes may be partitioned in racks. This is the hardware part of the infrastructure.
- YARN Infrastructure (Yet Another Resource Negotiator) is the framework responsible for providing the computational resources (e.g., CPUs, memory, etc.) needed for application executions.

Two important elements are:

- **Resource Manager** (one per cluster) is the master. It knows where the slaves are located (Rack Awareness) and how many resources they have. It runs several services, the most important is the Resource Scheduler which decides how to assign the resources.



- **Node Manager** (many per cluster) is the slave of the infrastructure.
- When it starts, it announces himself to the Resource Manager. Periodically, it sends an heartbeat to the Resource Manager.
- Each Node Manager offers some resources to the cluster.
- Its resource capacity is the amount of memory and the number of vcores.
- At run-time, the Resource Scheduler will decide how to use this capacity: a Container is a fraction of the NM capacity and it is used by the client for running a program.



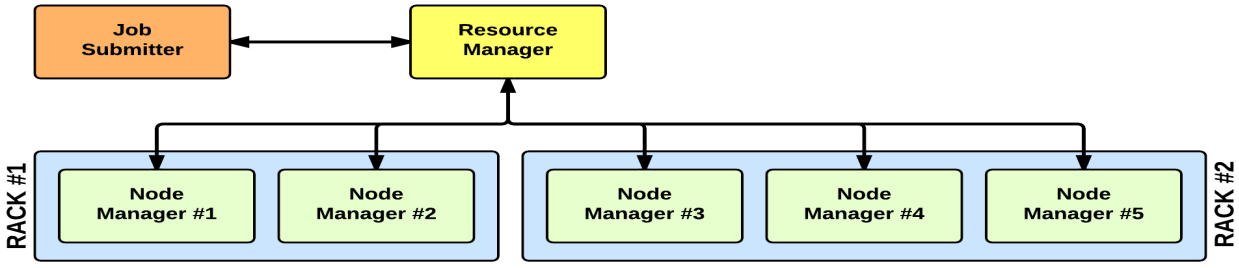
- HDFS Federation is the framework responsible for providing permanent, reliable and distributed storage. This is typically used for storing inputs and output (but not intermediate ones).
- Other alternative storage solutions. For instance, Amazon uses the Simple Storage Service (S3).
- The MapReduce Framework is the software layer implementing the MapReduce paradigm.

The YARN infrastructure and the HDFS federation are completely decoupled and independent:

First one provides resources for running an application while the second one provides storage.

The MapReduce framework is only one of many possible framework which runs on top of YARN (although currently is the only one implemented).

YARN: Application Startup

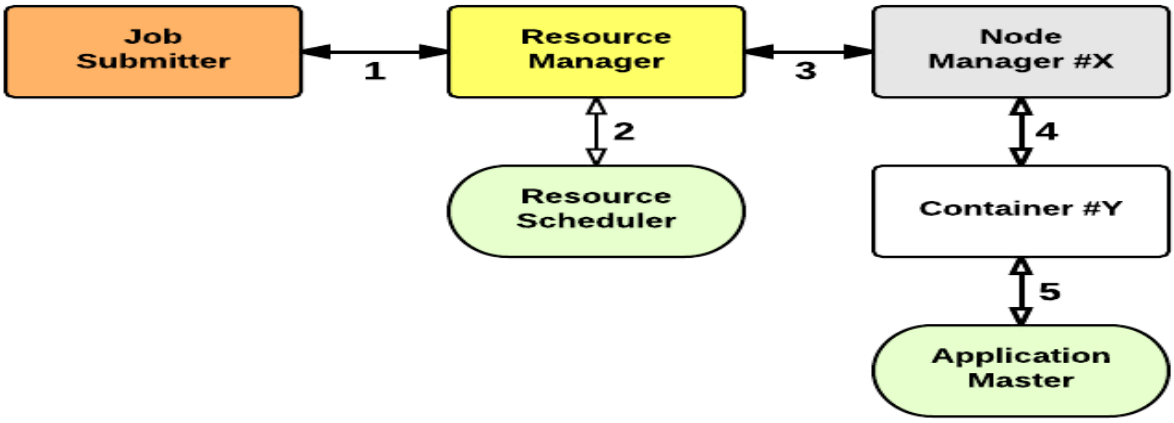


In YARN, there are at least three actors:

- the Job Submitter (the client)
- the Resource Manager (the master)
- the Node Manager (the slave)

The application startup process is the following:

1. a client submits an application to the Resource Manager
2. the Resource Manager allocates a container
3. the Resource Manager contacts the related Node Manager
4. the Node Manager launches the container
5. the Container executes the Application Master



The Application Master is responsible for the execution of a single application. It asks for containers to the Resource Scheduler (Resource Manager) and executes specific programs (e.g., the main of a Java class) on the obtained containers.

The Application Master knows the application logic and thus it is framework-specific.

The MapReduce framework provides its own implementation of an Application Master.

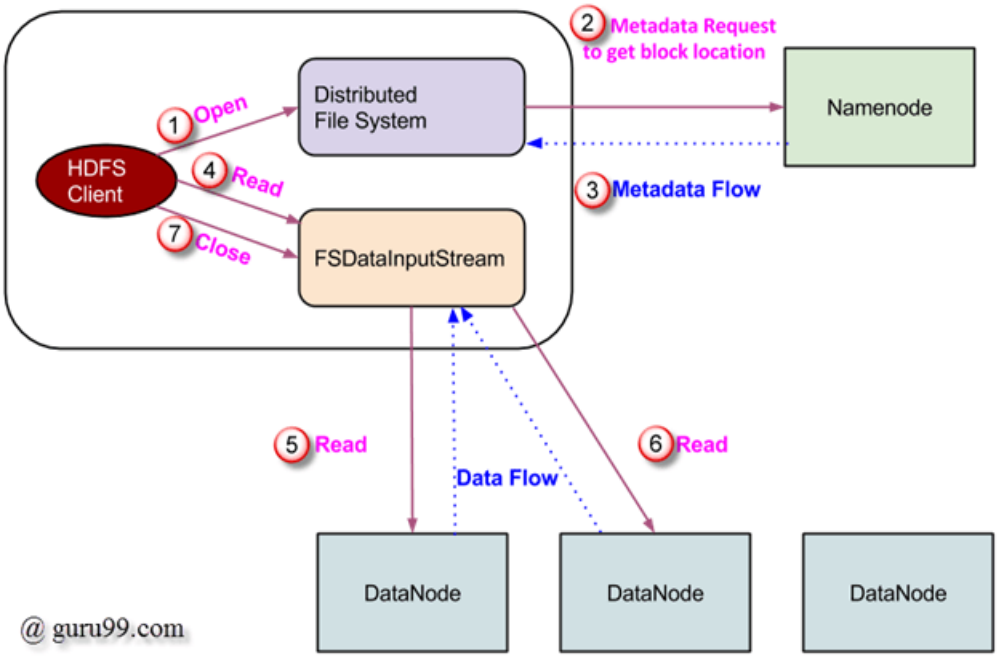
The Resource Manager is a single point of failure in YARN.

Using Application Masters, YARN is spreading over the cluster the metadata related to running applications.

This reduces the load of the Resource Manager and makes it fast recoverable.

Read Operation In HDFS

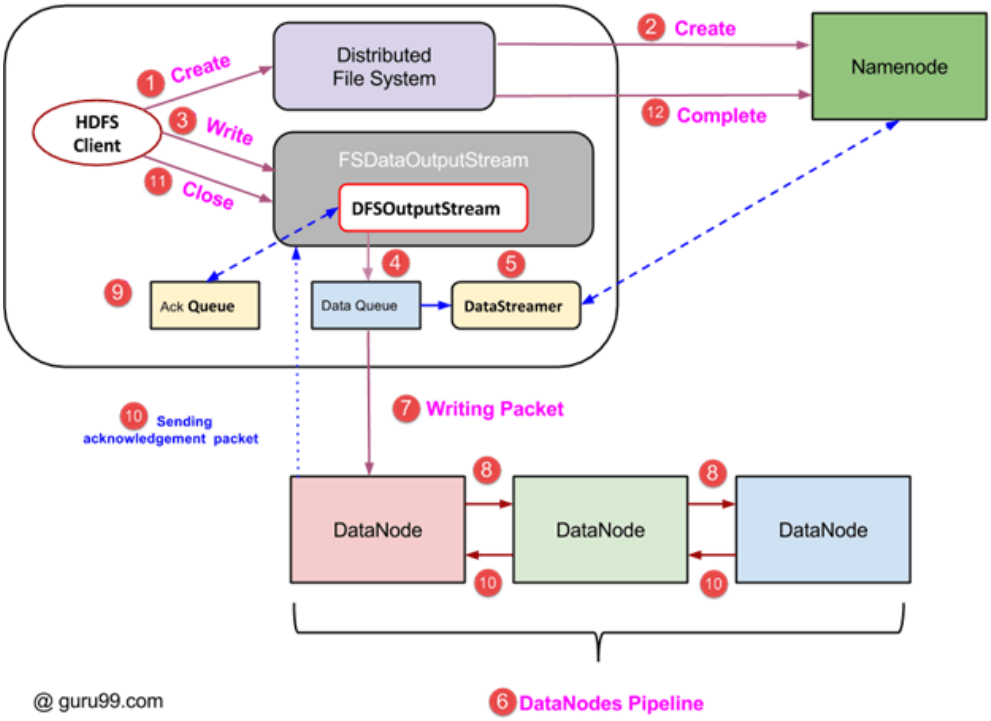
Data read request is served by HDFS, NameNode and DataNode. Let's call reader as a 'client'. Below diagram depicts file read operation in Hadoop.



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.



1. Client initiates write operation by calling 'create()' method of DistributedFileSystem object which creates a new file - Step no. 1 in above diagram.
2. Distributed FileSystem 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.

Hadoop HDFS Commands

Hadoop HDFS command are discussed below along with their usage, description, and examples.

Hadoop file system shell commands are used to perform various Hadoop HDFS operations and in order to manage the files present on HDFS clusters.

All the Hadoop file system shell commands are invoked by the bin/hdfs script

1. Create a directory in HDFS at given path(s).

Syntax: hadoop fs -mkdir<paths>

Example: hadoop fs -mkdir /user/saurzcode/dir1 /user/saurzcode/dir2

2. List the contents of a directory.

Usage :hadoop fs -ls <args>

Example: hadoop fs -ls /user/saurzcode

3. Upload and download a file in HDFS.

***Upload*:**

hadoop fs -put:

Copy single src file, or multiple src files from local file system to the Hadoop data file system

Syntax: hadoop fs -put <localsrc> ... <HDFS_dest_Path>

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

Download:

hadoop fs -get:

Copies/Downloads files to the local file system

Usage: hadoop fs -get <hdfs_src><localdst>

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

4. See contents of a file

Same as unix cat command:

Usage: hadoop fs -cat <path[filename]>

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

5. Copy a file from source to destination

This command allows multiple sources as well in which case the destination must be a directory.

```
Usage: hadoop fs -cp<source><dest>
Example: hadoop fs -cp /user/saurzcode/dir1/abc.txt /user/saurzcode/dir2
```

6. Copy a file from/To Local file system to HDFS

copyFromLocal

```
Usage: hadoop fs -copyFromLocal<localsrc> URI
Example: hadoop fs -copyFromLocal /home/saurzcode/abc.txt /user/saurzcode/abc.txt
```

Similar to put command, except that the source is restricted to a local file reference.

copyToLocal

```
Usage: hadoop fs -copyToLocal [-ignorecrc] [-crc] URI <localdst>
```

Similar to get command, except that the destination is restricted to a local file reference.

7. Move file from source to destination.

Note:- Moving files across filesystem is not permitted.

```
Usage :hadoop fs -mv <src><dest>
Example: hadoop fs -mv /user/saurzcode/dir1/abc.txt /user/saurzcode/dir2
```

8. Remove a file or directory in HDFS.

Remove files specified as argument. Deletes directory only when it is empty

```
Usage :hadoop fs -rm<arg>
 Example: hadoop fs -rm /user/saurzcode/dir1/abc.txt
```

Recursive version of delete.

Usage :hadoop fs -rmr<arg>

Example: hadoop fs -rmr /user/saurzcode/

9. Display last few lines of a file.

Similar to tail command in Unix.

Usage :hadoop fs -tail <path[filename]>

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

10. Display the aggregate length of a file.

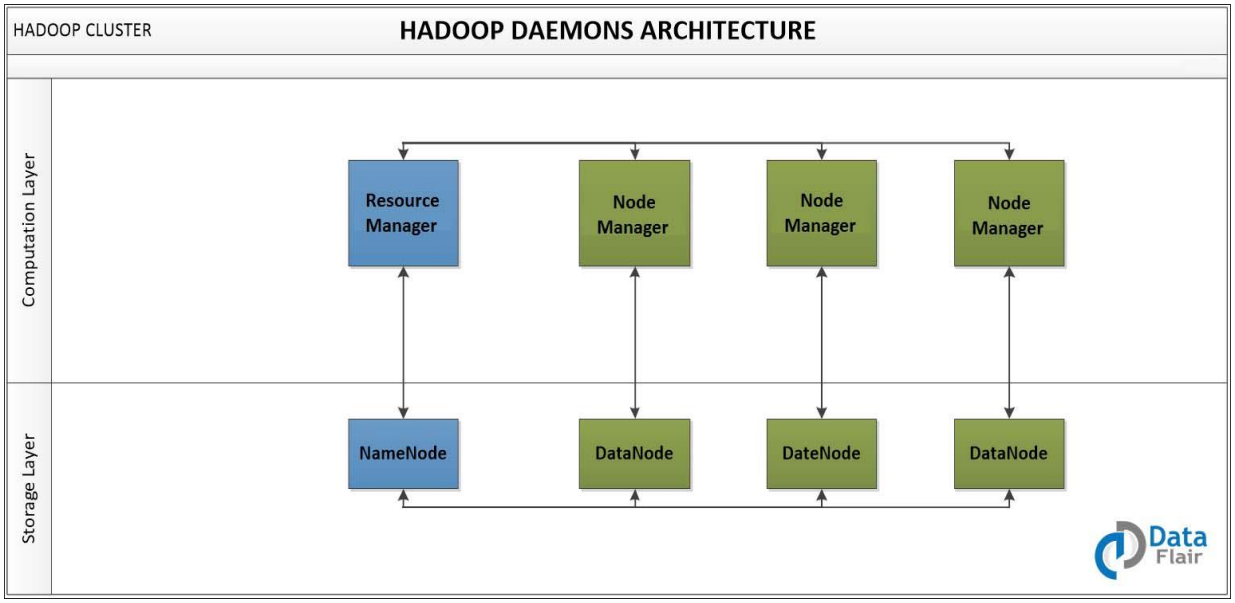
Usage :hadoop fs -du <path>

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

Please comment which of these commands you found most useful while dealing with Hadoop /HDFS

Hadoop Daemons

Daemons are the processes that run in the background. There are mainly 4 daemons which run for Hadoop.



- **Namenode** – It runs on master node for HDFS.
- **Datanode** – It runs on slave nodes for HDFS.
- **ResourceManager** – It runs on master node for Yarn.
- **NodeManager** – It runs on slave node for Yarn.

These 4 demons run for Hadoop to be functional. Apart from this, there can be secondary NameNode, standby NameNode, Job HistoryServer, etc.

Hadoop Flavors

Below are the various flavors of Hadoop.

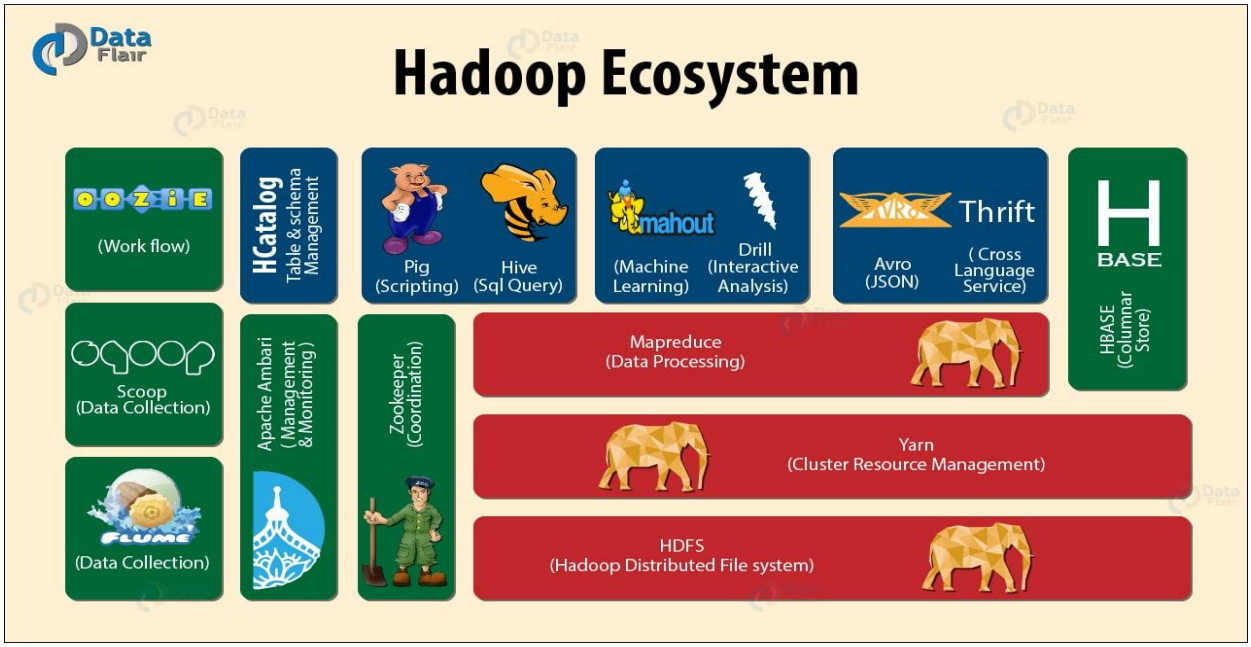
- **Apache** – Vanilla flavor, as the actual code is residing in Apache repositories.
- **Hortonworks** – Popular distribution in the industry.
- **Cloudera** – It is the most popular in the industry.
- **MapR** – It has rewritten HDFS and its HDFS is faster as compared to others.
- **IBM** – Proprietary distribution is known as Big Insights.

All the databases have provided native connectivity with Hadoop for fast data transfer. Because, to transfer data from Oracle to Hadoop, you need a connector.

All flavors are almost same and if you know one, you can easily work on other flavors as well.

Hadoop Ecosystem

In this section of Hadoop tutorial, we will cover Hadoop ecosystem components. Let us see what all the components form the **Hadoop Eco-System**:



- **Hadoop HDFS** – Distributed storage layer for Hadoop.
- **Yarn** – Resource management layer introduced in Hadoop 2.x.
- **Hadoop Map-Reduce** – Parallel processing layer for Hadoop.
- **HBase** – It is a column-oriented database that runs on top of HDFS. It is a NoSQL database which does not understand the structured query. For sparse data set, it suits well.
- **Hive** – Apache Hive is a data warehousing infrastructure based on Hadoop and it enables easy data summarization, using SQL queries.
- **Pig** – It is a top-level scripting language. As we use it with Hadoop. Pig enables writing complex data processing without Java programming.
- **Flume** – It is a reliable system for efficiently collecting large amounts of log data from many different sources in real-time.
- **Sqoop** – It is a tool design to transport huge volumes of data between Hadoop and RDBMS.
- **Oozie** – It is a Java Web application uses to schedule Apache Hadoop jobs. It combines multiple jobs sequentially into one logical unit of work.
- **Zookeeper** – A centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.
- **Mahout** – A library of scalable machine-learning algorithms, implemented on top of Apache Hadoop and using the MapReduce paradigm.