Hadoop Core Components

Hadoop Core Components

powered by Apache hadoop = hadoop HDFS + hadoop Map Reduce

Hadoop Core Components

HADOOP has two main components

- ✓ **HDFS-Hadoop Distributed File System(Storage)**
  - ✓ Distributed across nodes
  - ✓ Natively redundant
  - ✓ Name node tracks locations

- ✓ **Map Reduce (Processing)**
  - ✓ Splits a task across processors
  - ✓ “near” the data & assemble results
  - ✓ Self-healing, Hand bandwidth
  - ✓ Clustered storage
  - ✓ Job tracker manages the task tracker

HDFS(Storage) - Demons

Name Node (FSIMAGE, EDIT LOG)

Data Node

Secondary Name Node

(FSIMAGE)

Map Reduce (Processing) - Demons

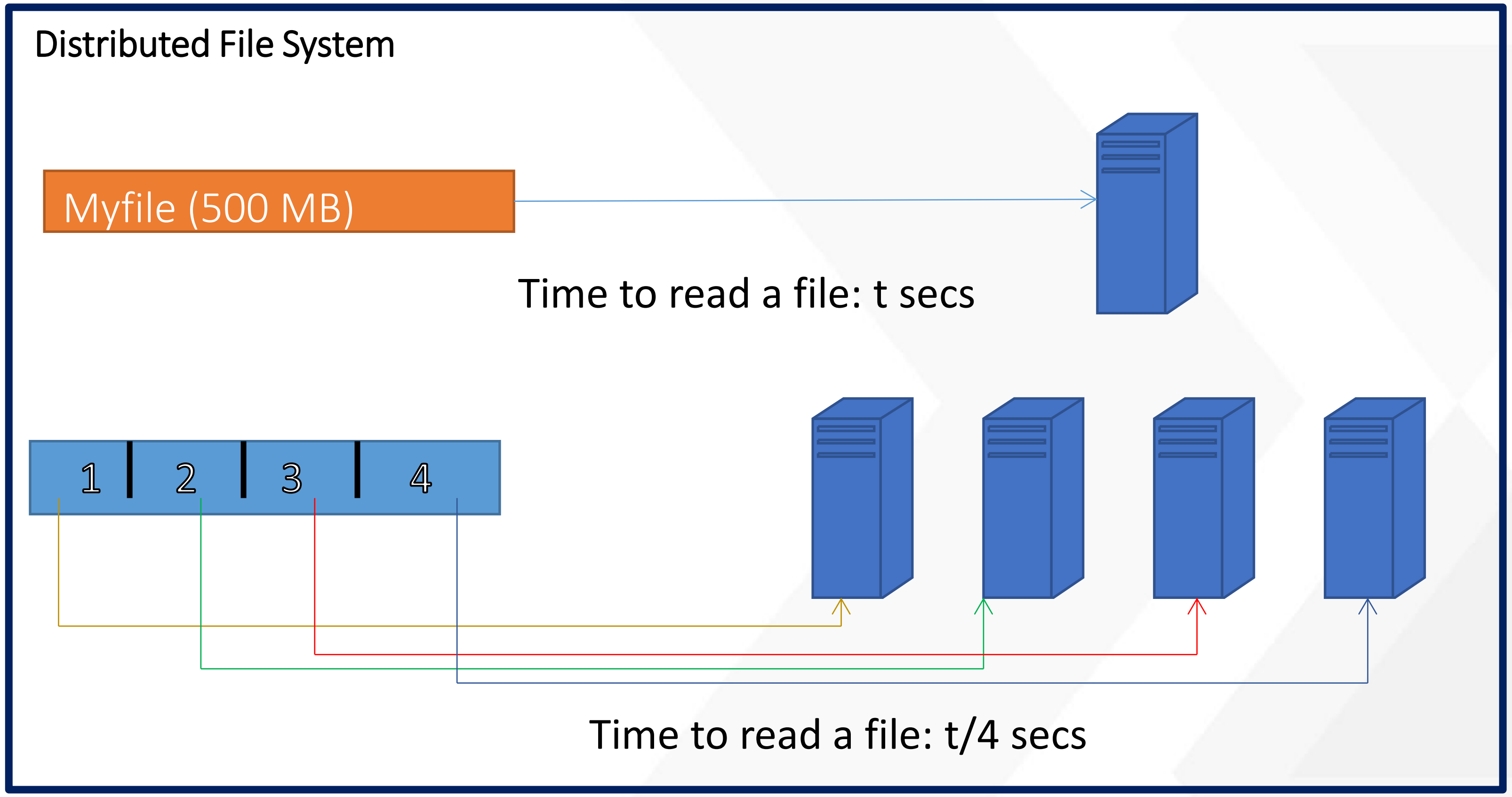
Resource Manager (job tracker)

Node Manager (task tracker)

Resource Manager, Node manager are the daemons of YARN(popularly

Hadoop - HDFS

Distributed File System

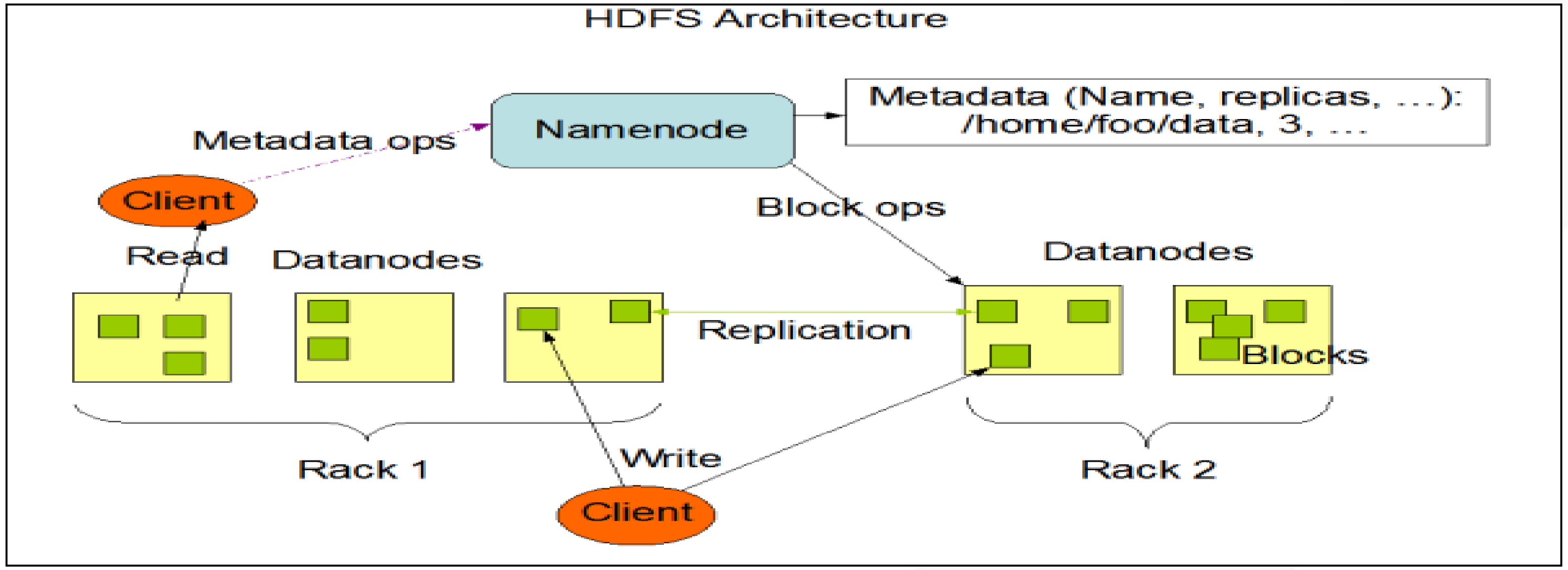


ANALYTIXLABS

Hadoop Distributed File System (HDFS)

What is HDFS?

- ✓ *HDFS is derived from concepts of* ***Google file system****.*
- ✓ *An important characteristic of Hadoop is* ***partitioning of data*** *and execution of application* ***computations in parallel****, close to their data.*
- ✓ *On HDFS, data files are* ***replicated*** *as sequences of blocks in the cluster*
- ✓ *A Hadoop cluster scales computation capacity and storage capacity by simply adding commodity servers*
- ✓ HDFS is the default distributed file system used by the Hadoop MapReduce computation



Key HDFS Features

- Runs with commodity hardware
- Master slave paradigm
- Distributed
- Scalable (scale out Architecture)
- Secured
- Reliable (Tunable Replication)
- Fault Tolerance
- High-throughput
- Small number of large files vs. large number of small files
- Write Once Read Many Times
- Sequential/Streaming Access

Regular File System vs. HDFS

Regular File System

- Each block of data is small in size; approximately 4K bytes.
- All the blocks of a file is stored on the local disk of a computer system.
- Storage Capacity of system should be large enough to store the file.
- Large data access suffers from disk I/O problems; mainly because of multiple seek operation.

HDFS

- Each block of data is very large in size; 64MB/128MB by default.
- Different blocks of a file are stored on different nodes.
- A large file is divided into blocks, even if storage capacity of a system is not too large, save file into cluster.
- Reads huge data sequentially after a single seek

HDFS Architecture

- HDFS is a java-based file system and is the place where all the data in the Hadoop cluster resides.
- HDFS has been restructured in the second version of Hadoop to support multiple types of data processing units.

As of now we have two versions in Hadoop;Hadoop1.x and Hadoop 2.x.

Hadoop1.x:
This is the very first version of Hadoop built to handle BigData. HDFS and MapReduce are the two steps involved in processing the data in Hadoop1.x architecture. It process data in Batches, hence the name ‘Batch processing’.

Hadoop2.x
Hadoop2.x is the enhanced version of Hadoop1.x. This version has introduced to overcome the problems and shortcomings of Hadoop1.x. A new feature called YARN(Yet Another Resource negotiator) has been introduced in Hadoop2.x. Here HDFS is used along with YARN to process the data. With YARN, Hadoop is able to process different forms of data. Along with Hadoop, we can use many other tools to process the data.

HDFS Architecture

In Hadoop1.x the components of HDFS:

- ✓NameNode
- ✓DataNode
- ✓Job Tracker
- ✓Task Tracker
- ✓Secondary NameNode

✓NameNode, DataNode, Secondary NameNode are the daemons of HDFS

✓Job tracker, Task tracker are the daemons of Map reduce

In Hadoop2.x the components of HDFS:

- ✓NameNode
- ✓DataNode
- ✓Resource Manager
- ✓Node Manager
- ✓Secondary NameNode

✓NameNode, DataNode, Secondary NameNode are the daemons of HDFS

✓Resource Manager, Node manager are the daemons of YARN(popularly known as the Map reduce 2.0)

Along with Hadoop, we can use many other tools to process the data. They are as follows:

HDFS+MapReduce- Batch processing
HDFS+Storm – Real-time processing
HDFS+Spark – Near real-time processing

HDFS Daemons

NameNode:
NameNode holds the meta data for the HDFS like Namespace information, block information etc. When in use, all this information is stored in main memory. But these information also stored in disk for persistence storage.

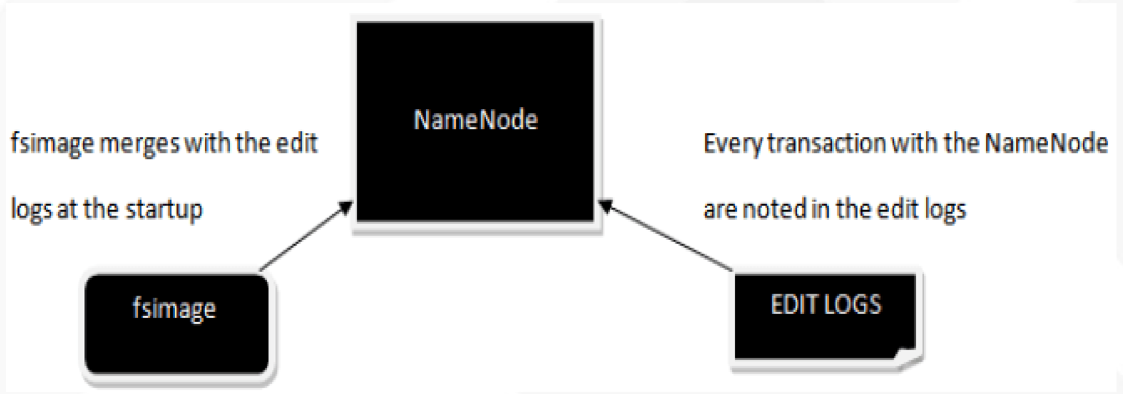
There are two different files associated with the NameNode

Fsimage – It is the snapshot of the filesystem

Edit logs – After the start of NameNode it maintains the every transaction that happened with NameNode.

The NameNode maintains the namespace tree and the mapping of blocks to DataNodes.
The Client communicates with the NameNode and provides data to the HDFS through it. The HDFS then stores data as blocks inside DataNodes.

By default, the block size in a hadoop cluster is 128MB. NameNode maintains the meta data information of all the blocks present inside the hadoop cluster like permissions, modification and access times, namespace and disk space quotas . This is the reason NameNode is also called as the Master node.

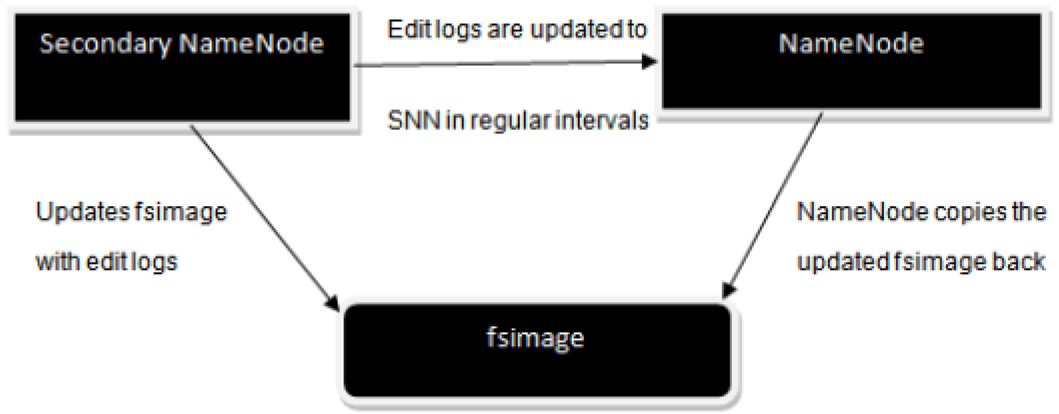

NameNode
fsimage merges with the edit logs at the startup
Every transaction with the NameNode are noted in the edit logs
fsimage
EDIT LOGS

HDFS Demons

Secondary Name Node:
It asks the NameNode for its edit logs in regular intervals and copies them into the fsimage.

After updating the fsimage, NameNode copy back that fsimage. NameNode uses this fsimage when it starts, this eventually will reduce the startup time.

The main theme of secondary NameNode is to maintain a checkpoint in HDFS. When a failure occurs SNN won't become NameNode it just helps NameNode in bringing back its data. SNN is also called as checkpoint node in hadoop's architecture.



HDFS Demons

DataNode:

This is the place where actual data is stored in the hadoop cluster in a distributed manner.
Every DataNode will have a block scanner and it directly reports to the NameNode about the blocks which it is handling.

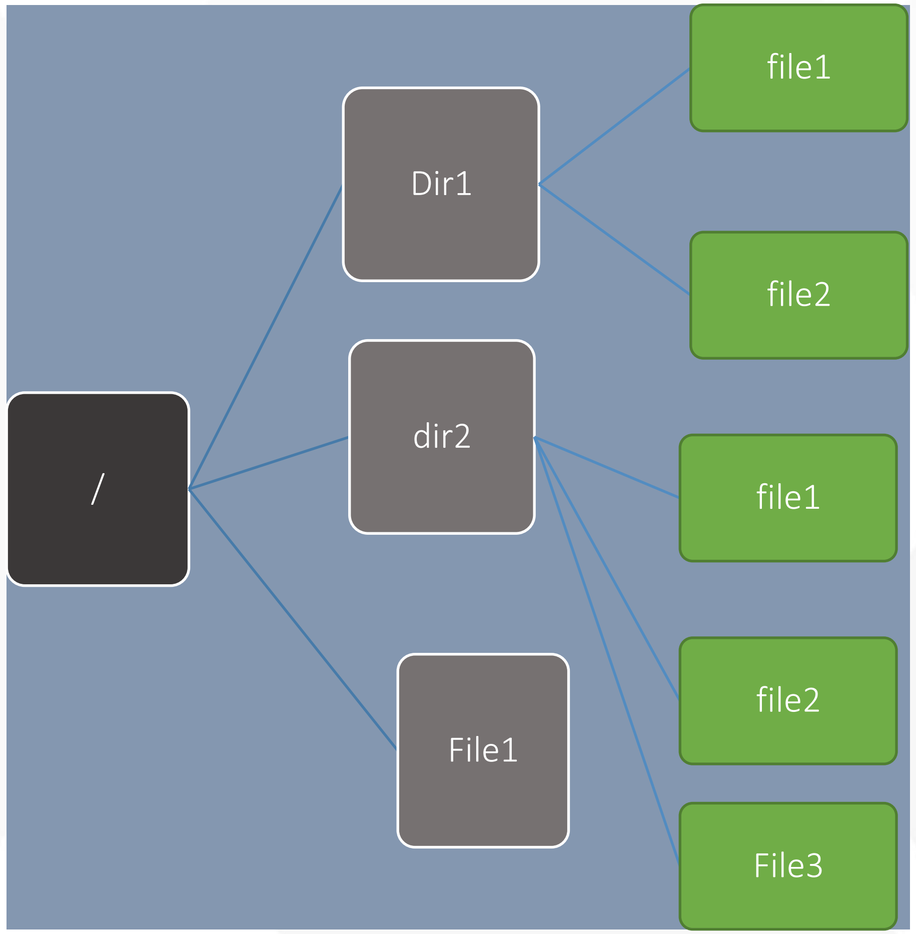
The DataNodes communicate with the NameNode by sending Heartbeats for every 3seconds and if the NameNode does not receive any Heartbeat for 10 minutes, then it treats the DataNode as a dead node and re-replicates the blocks.

During startup each DataNode connects to the NameNode and performs a handshake. The purpose of the handshake is to verify the namespace ID and the software version of the DataNode. If either does not match that of the NameNode, the DataNode automatically shuts down.

The namespace ID is assigned to the file system instance when it is formatted. The namespace ID is persistently stored on all nodes of the cluster.

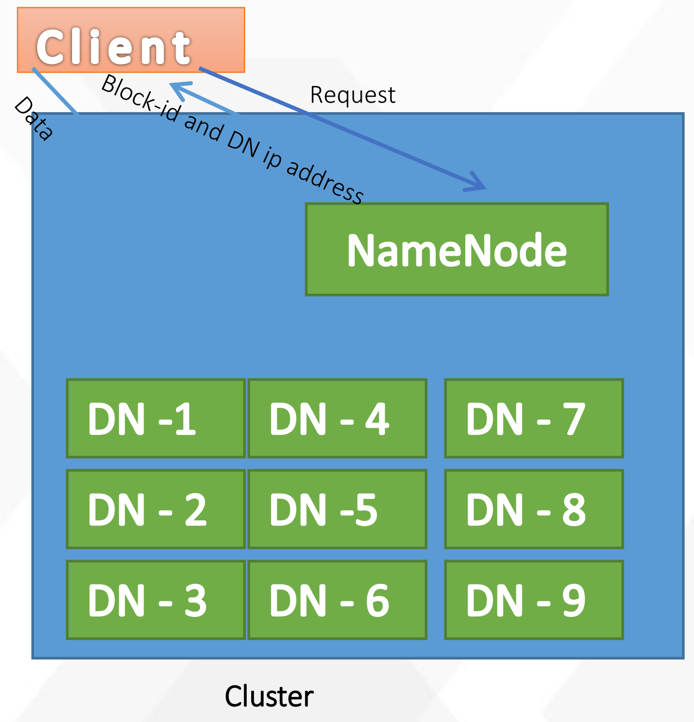
HDFS Name Space

- Hierarchical file system
- Directories and Files
- Sits on the native linux file system
- Uses same commands as linux file system
- Create, move, copy, rename or delete
- NameNode maintains the HDFS Name space.
- Any changes in HDFS Name space is recorded in the metadata stored in RAM of NameNode.
- Single Name Space for the cluster.



Client Interaction with cluster

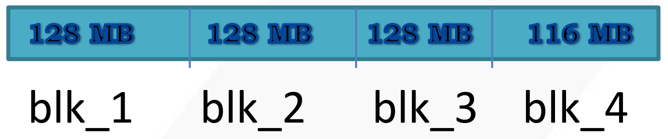
- Client requests NameNode to write a file in cluster.
- NameNode checks the authenticity of client.
- No replacement strategy, so NameNode checks if file already exists in cluster. Sends error if file exists.
- Finds the file size and divide the file into chunks based upon block-size called blocks.
- NameNode gives unique block-id to each block and decides on which DataNode to store which block.
- Sends block-id and DataNode ip address to client one by one.
- Client writes data directly to DataNode.



Data Blocks

- Default block size is 64 MB/ 128 MB, but configurable.
- Each file is divided into one or more blocks depending upon file size.
- All the blocks except the last one contains data equal to block-size.
- Each block has a unique block-id. (Integer)
- Blocks are stored on DataNodes.
- Blocks on the DataNodes are stored as complete and independent files.

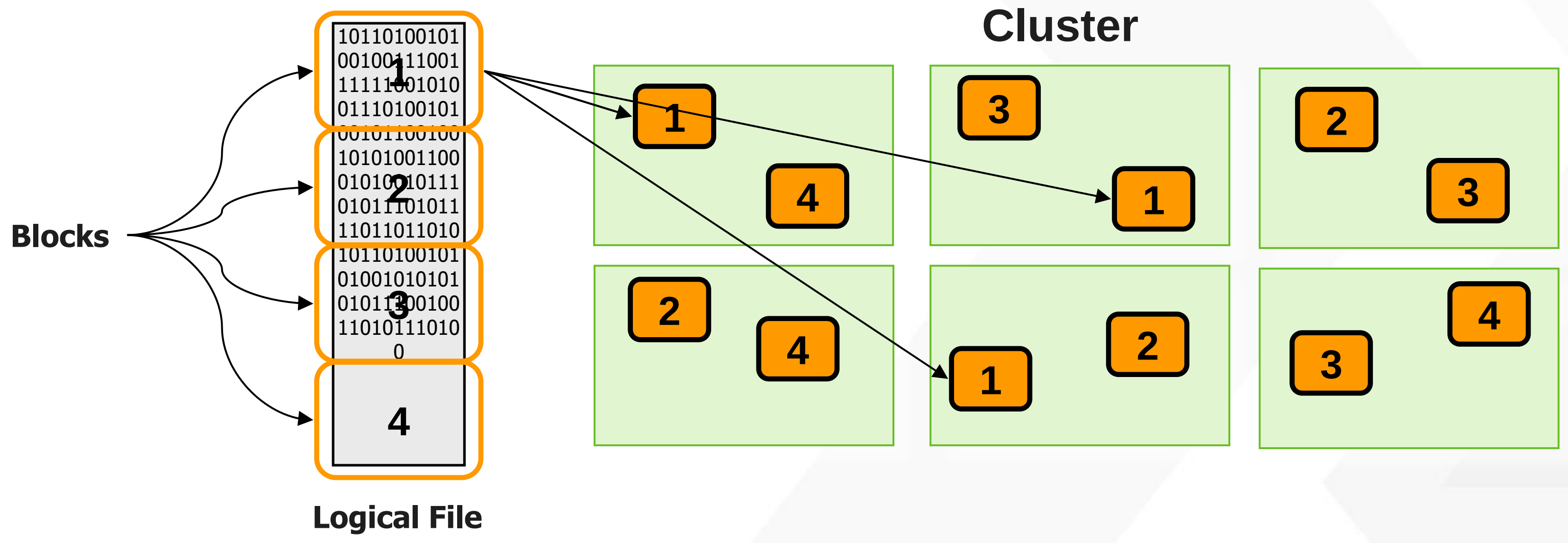
Myfile (500 MB) divided into 4 blocks
(1,2,3 and 4)



HDFS: Reliability

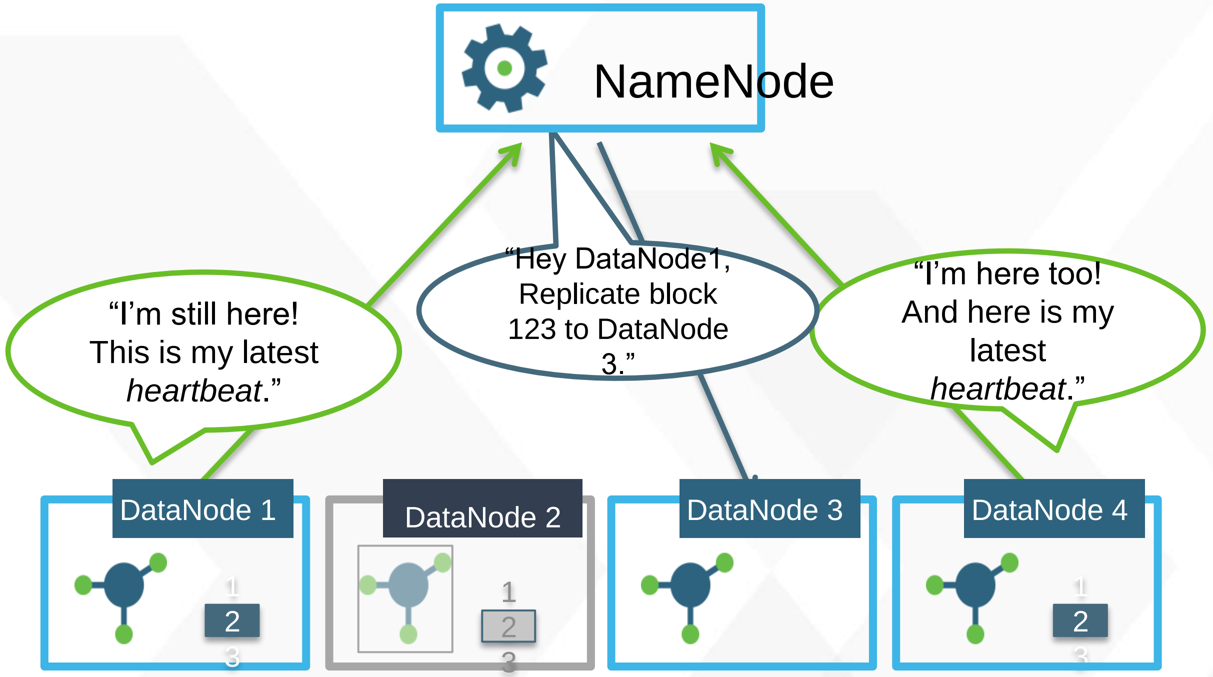
Fault Tolerant Distributed Storage

- Divide files into big blocks and distribute 3 copies **randomly** across the cluster
- Processing Data Locality
  - Not Just storage but computation



HDFS: Fault Tolerance

- Hadoop Cluster is built-up of commodity hardware.
- Frequent failure is norm rather than exception.
- All the DataNodes send heartbeat to NameNode after every minute to indicate that they are alive.
- After every 10th heartbeat, each DataNode sends block report indicating which blocks are available.



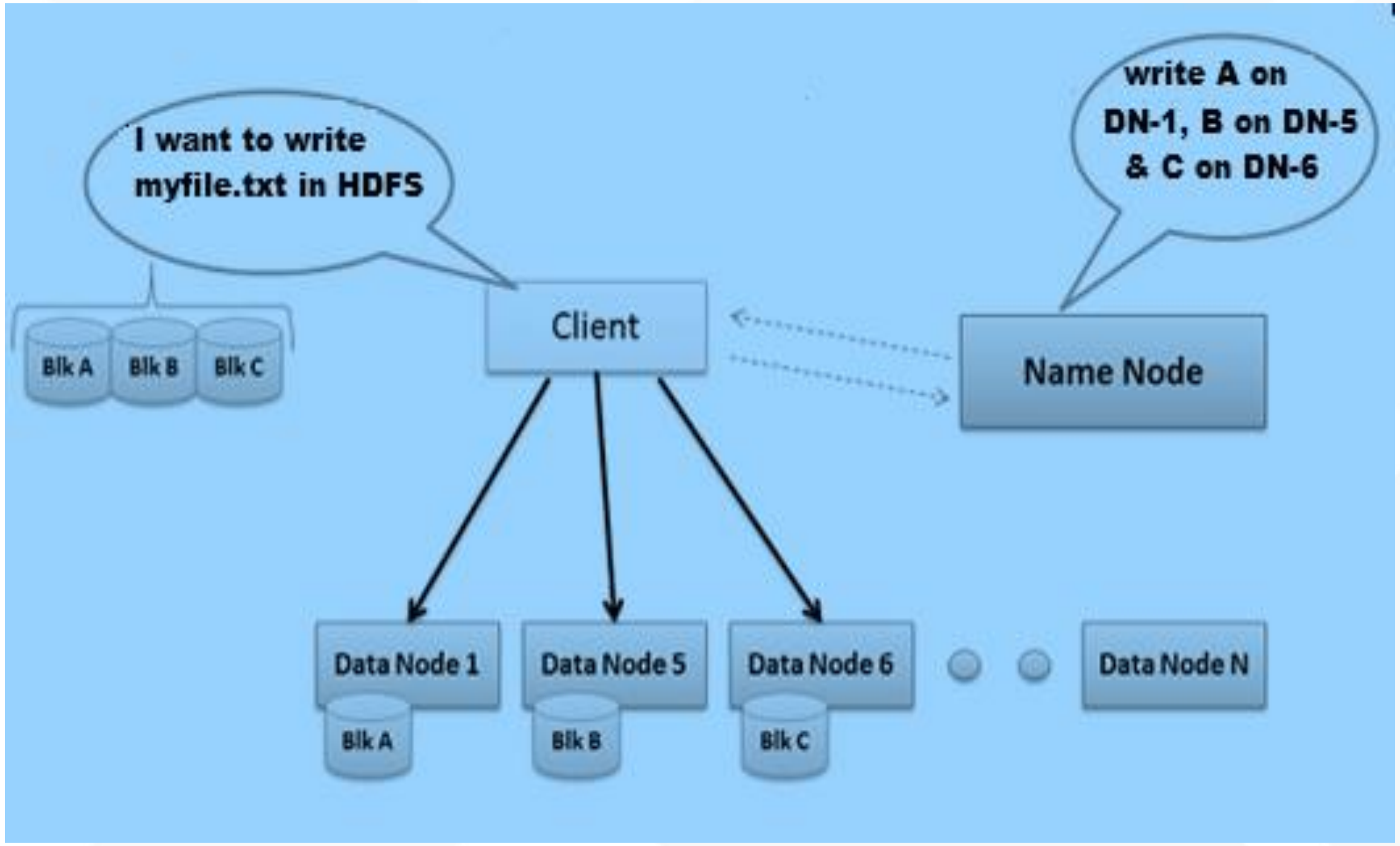
NameNode
"I'm still here! This is my latest *heartbeat*."
"Hey DataNode1, Replicate block 123 to DataNode 3."
"I'm here too! And here is my latest *heartbeat*."
DataNode 1
DataNode 2
DataNode 3
DataNode 4
1
2
3

- NameNode does not get heartbeat from a DataNode.
- Waits to get the block report.
- If NameNode does not get block report, assumes that DataNode is not live.
- Replicates the blocks of that DataNode to some other DataNodes.

ANALYTIXLABS

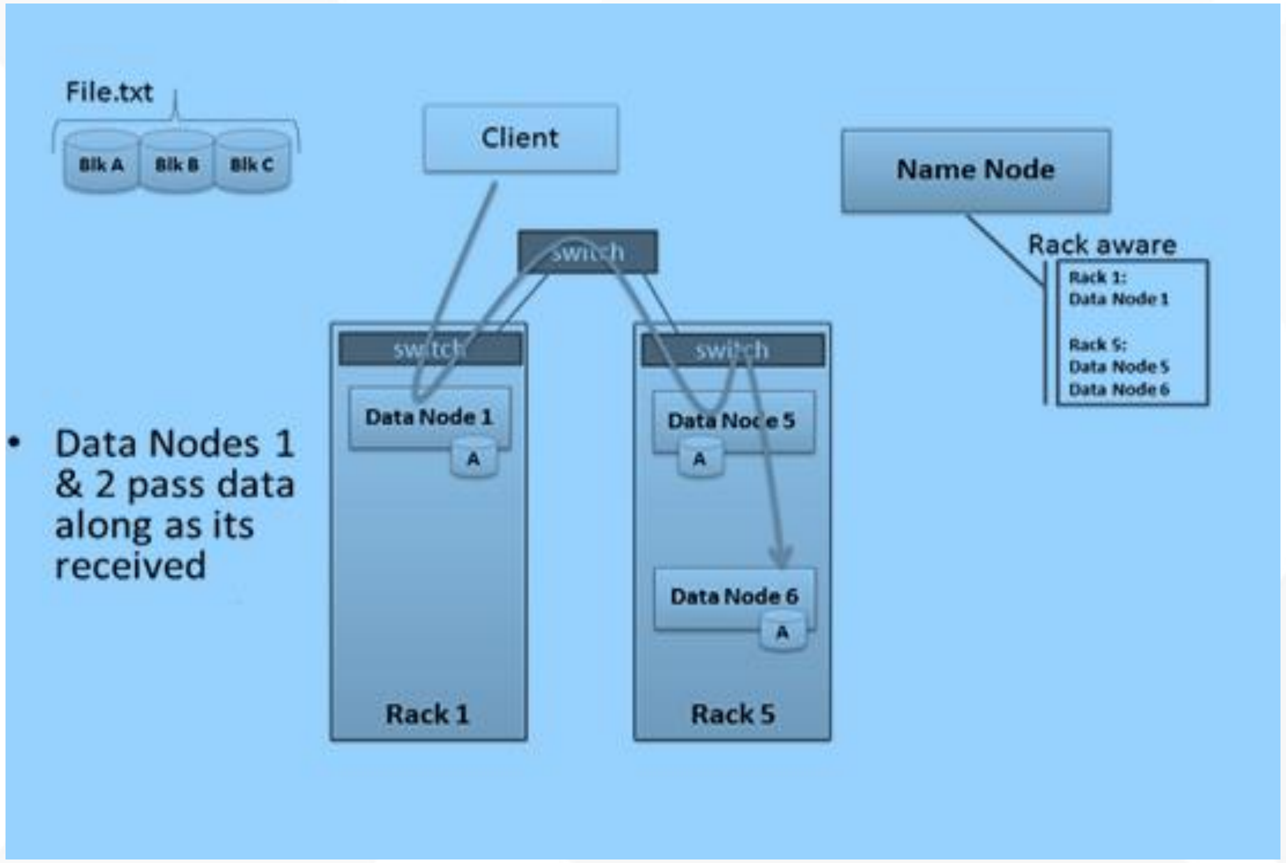
Writing File to HDFS

- Client sends a request to NameNode.
- Namenode sends Block-Id alongwith IP address of DataNode on which to write block.
- Client writes block data directly to DataNode.
- Client repeats the process for next block.



Pipelined Write to HDFS

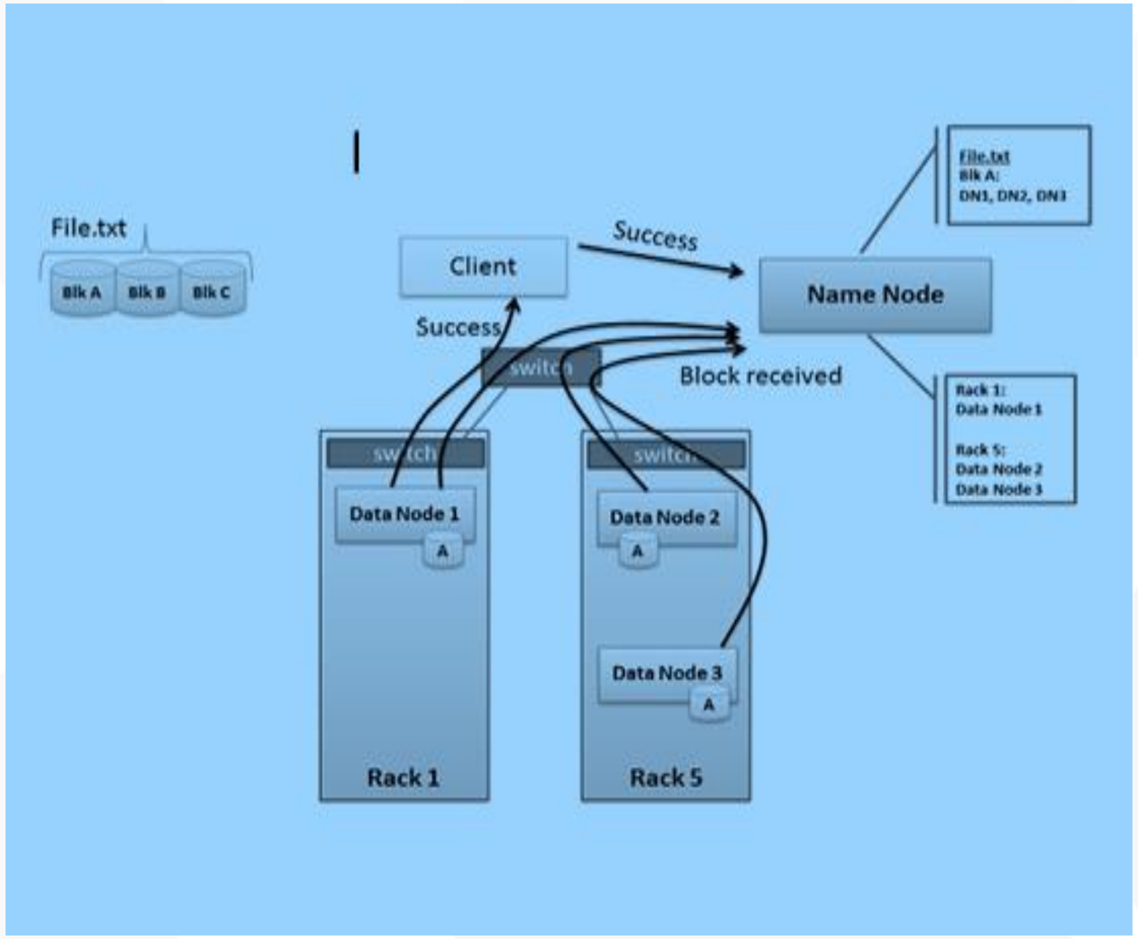
- Client sends a request to first DataNode to create a pipeline with other DataNodes.
- Once pipeline is created, client writes data to first DataNode.
- First DataNode writes and forwards data to next one and so on.
- All the DataNodes send acknowledgement to NameNode.
- Client repeats the process for next block.



- Data Nodes 1 & 2 pass data along as its received

Meta Data

- NameNode maintains information about each file and block in metadata.
- Metadata is kept in the RAM of NameNode.
- In the metadata, information about files such as filename, owner, group, creationTime, accessTime, size of file, block size and replication factor along with block-Ids belonging to that file.
- Location of the blocks are not kept in metadata.

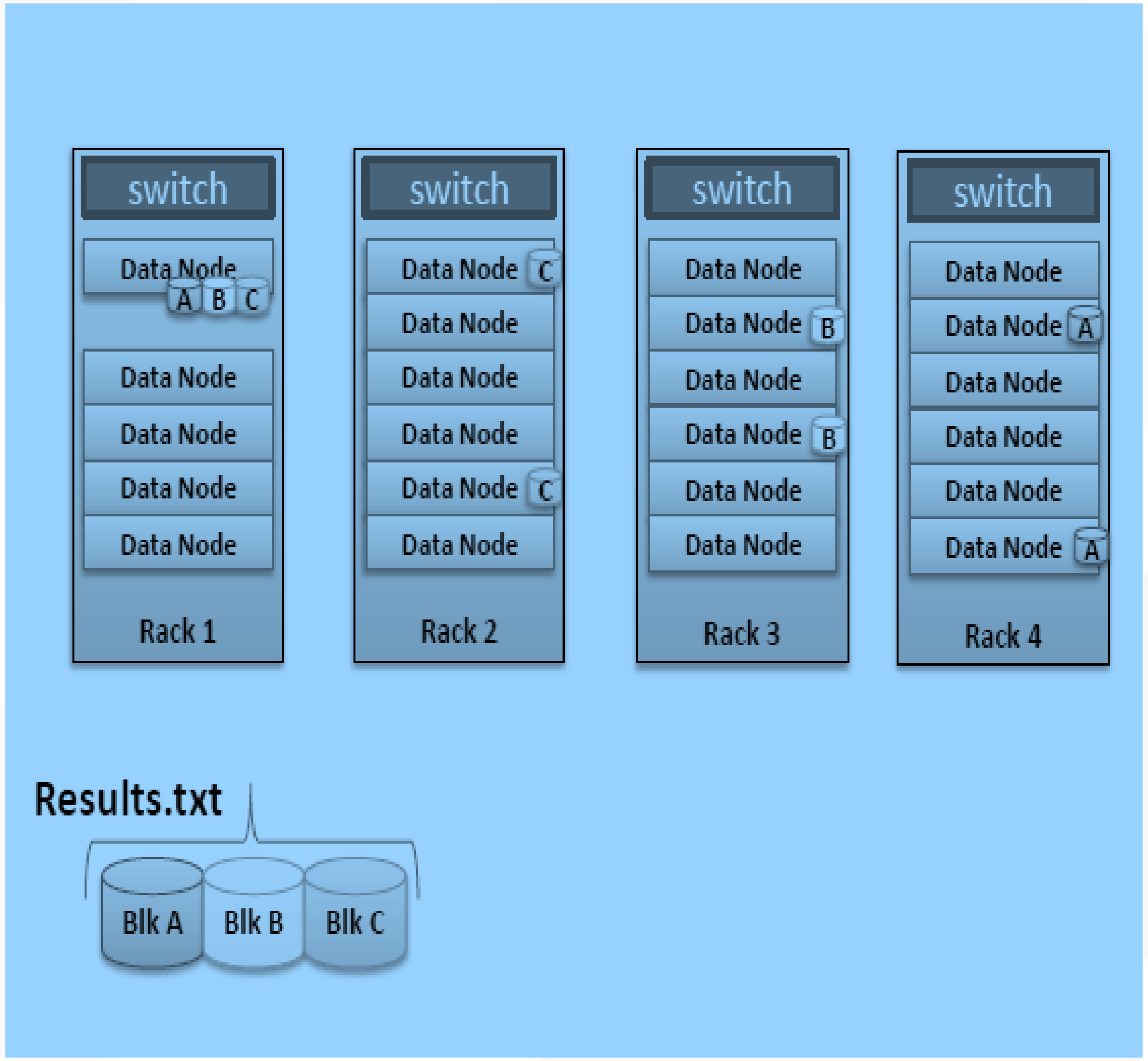


ANALYTIXLABS

Data Replication Topology

By default, Hadoop places block replicas as:

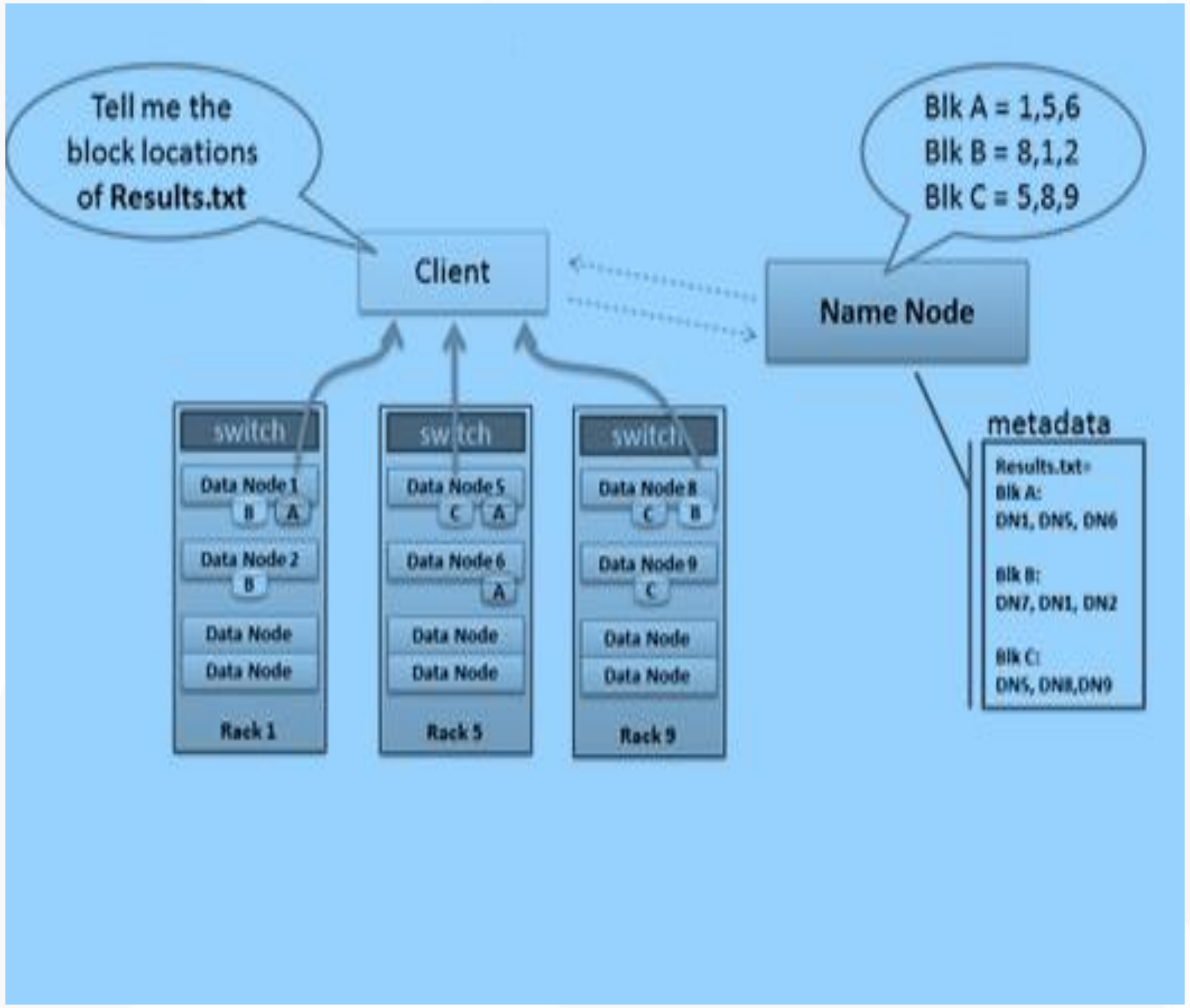
- 1st replica is placed on one node of any rack.
- 2nd replica is placed on some node of different rack.
- 3rd replica is placed on different machine on the same rack as of 2nd replica.



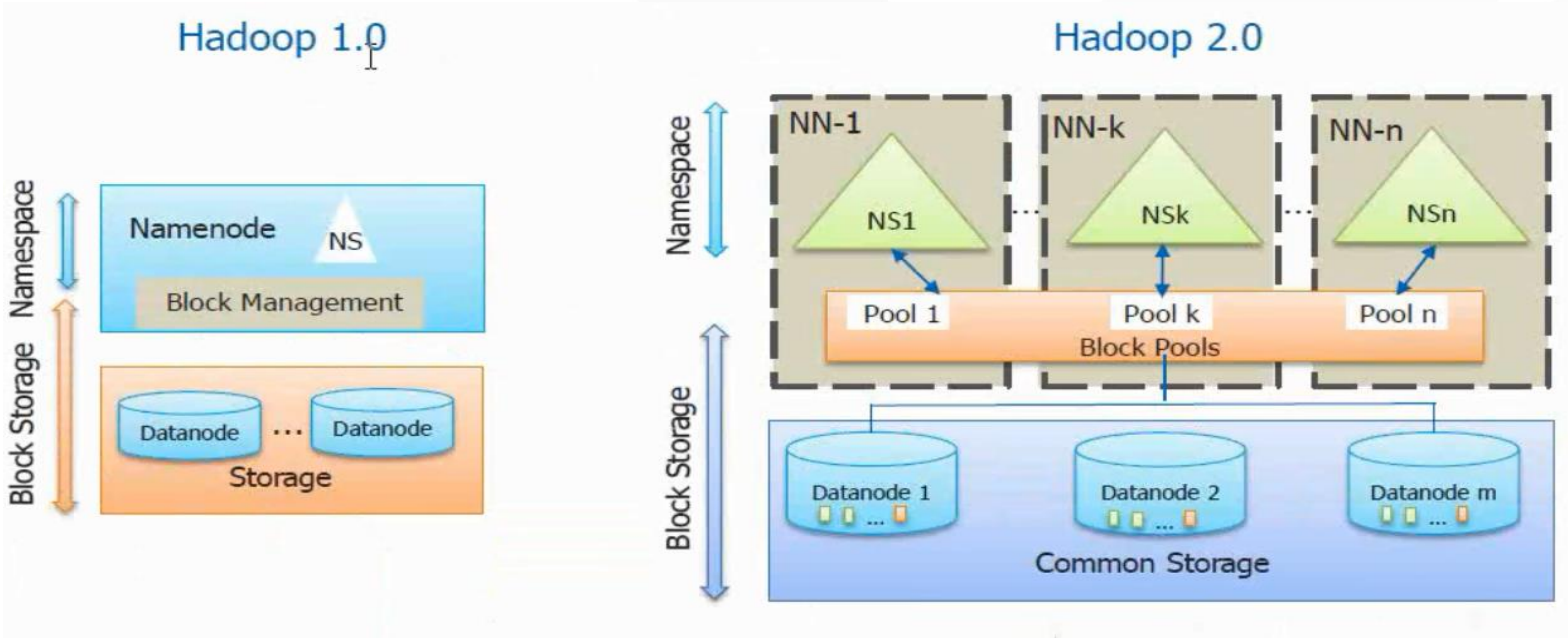
ANALYTIXLABS

Data Reading from HDFS

- Client sends the request to NameNode.
- NameNode sends the list of IP addresses of all DataNodes on which block exists.
- Client interacts with the first DataNode in the list on which block exists.
- Client reads the data directly from the DataNode.



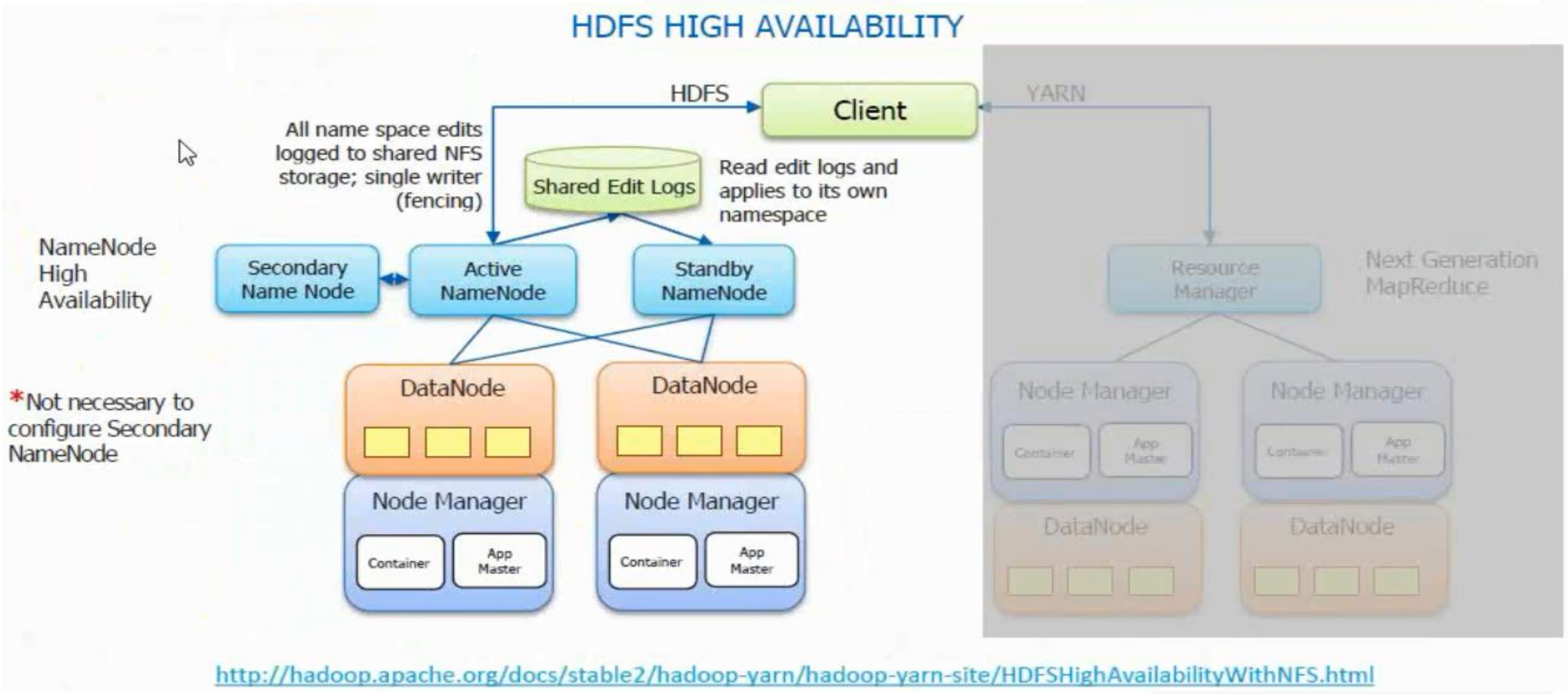
Hadoop 2.x Architecture - Federation



http://hadoop.apache.org/docs/stable2/hadoop-project-dist/hadoop-hdfs/Federation.html

ANALYTIXLABS

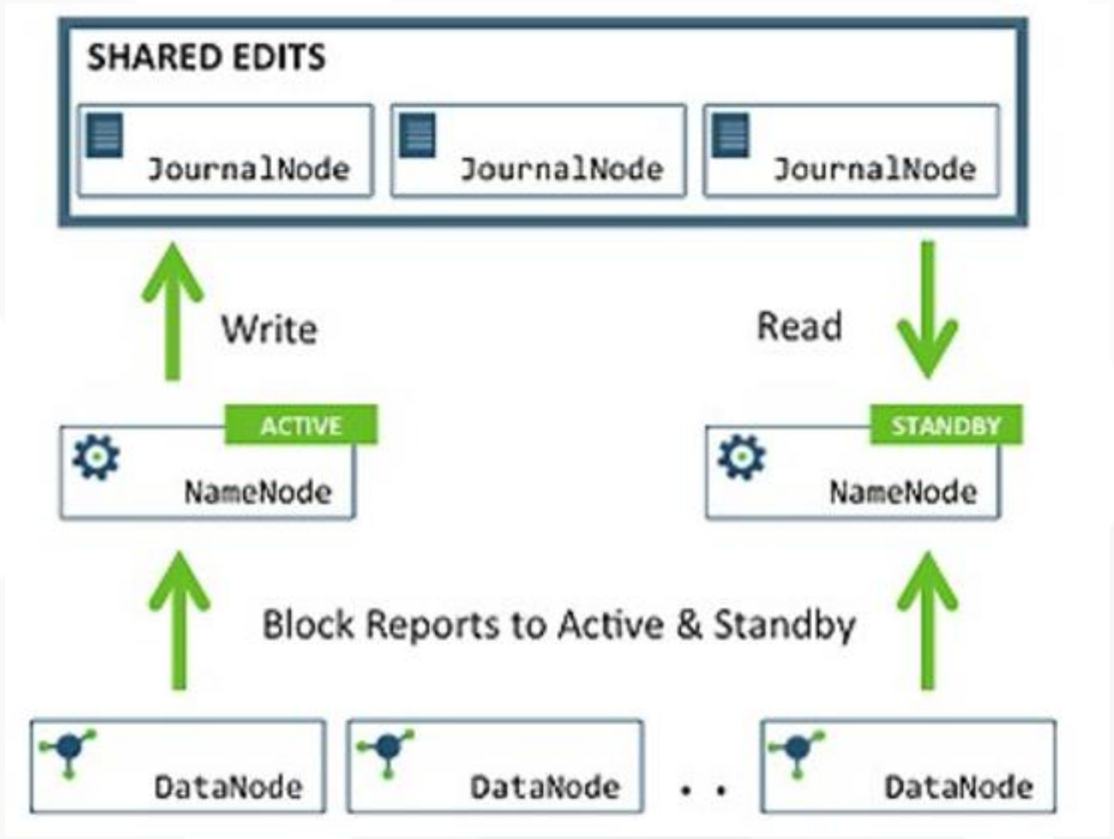
HDFS High Availability feature with shared edit log


HDFS HIGH AVAILABILITY
HDFS
Client
YARN
All name space edits logged to shared NFS storage; single writer (fencing)
Shared Edit Logs
Read edit logs and applies to its own namespace
NameNode High Availability
Secondary Name Node
Active NameNode
Standby NameNode
Resource Manager
Next Generation MapReduce
*Not necessary to configure Secondary NameNode
DataNode
DataNode
Node Manager
Node Manager
Container
App Master
Container
App Master
Node Manager
Node Manager
Container
App Master
Container
App Master
DataNode
DataNode
http://hadoop.apache.org/docs/stable2/hadoop-yarn/hadoop-yarn-site/HDFSHighAvailabilityWithNFS.html

ANALYTIXLABS

HDFS High Availability using Quoram Journal Manager

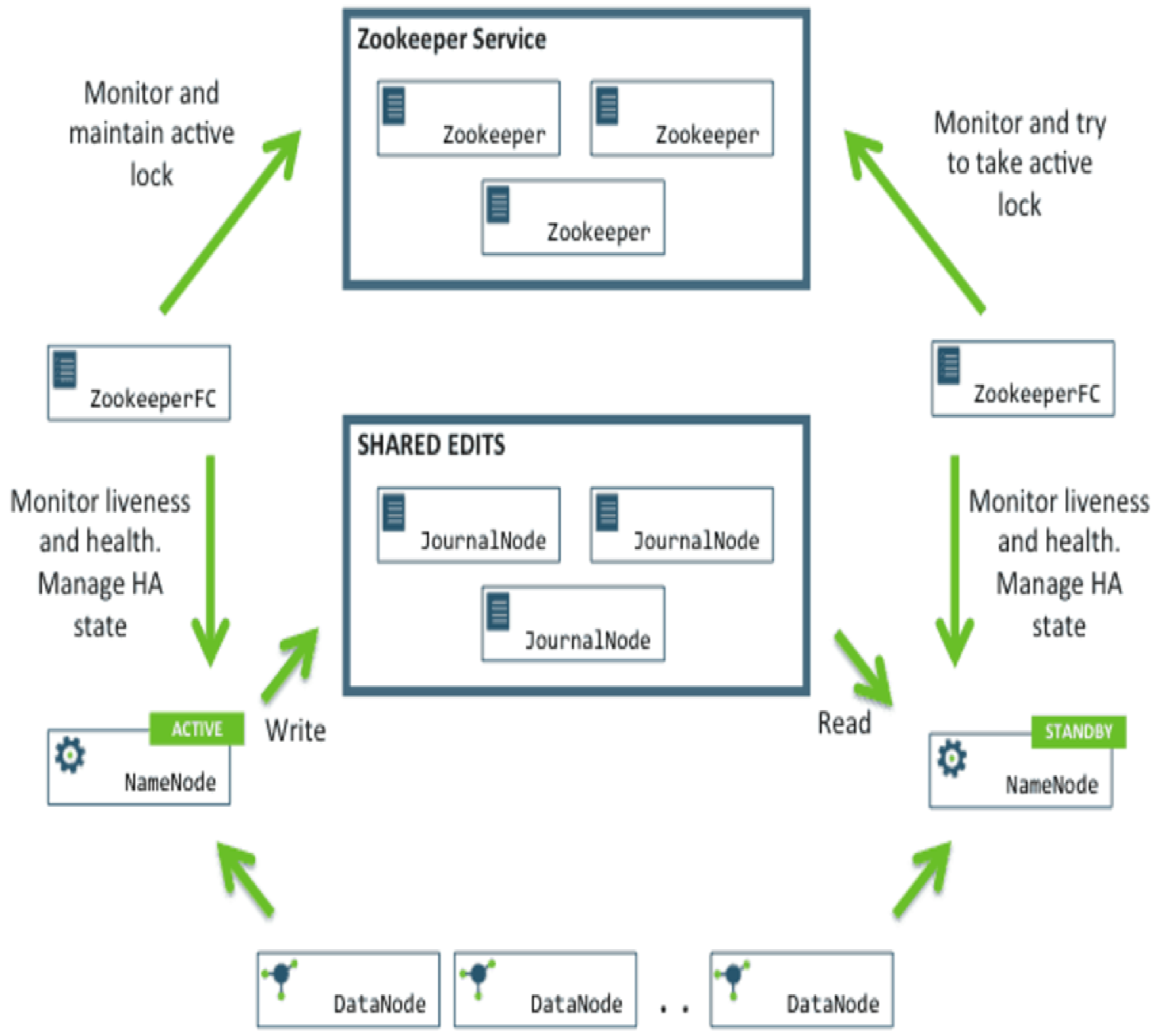
- QJM is dedicated HDFS implementation
- QJM runs as a group of Journal nodes and each edit must be written to majority of journal nodes
- QJM only allows one Name Node to write to the edit log at one time
- QJM has ssh fencing method implemented which helps to avoid the Name Node split-brain scenario



HDFS HA & Automatic failover using QJM and ZooKeeper

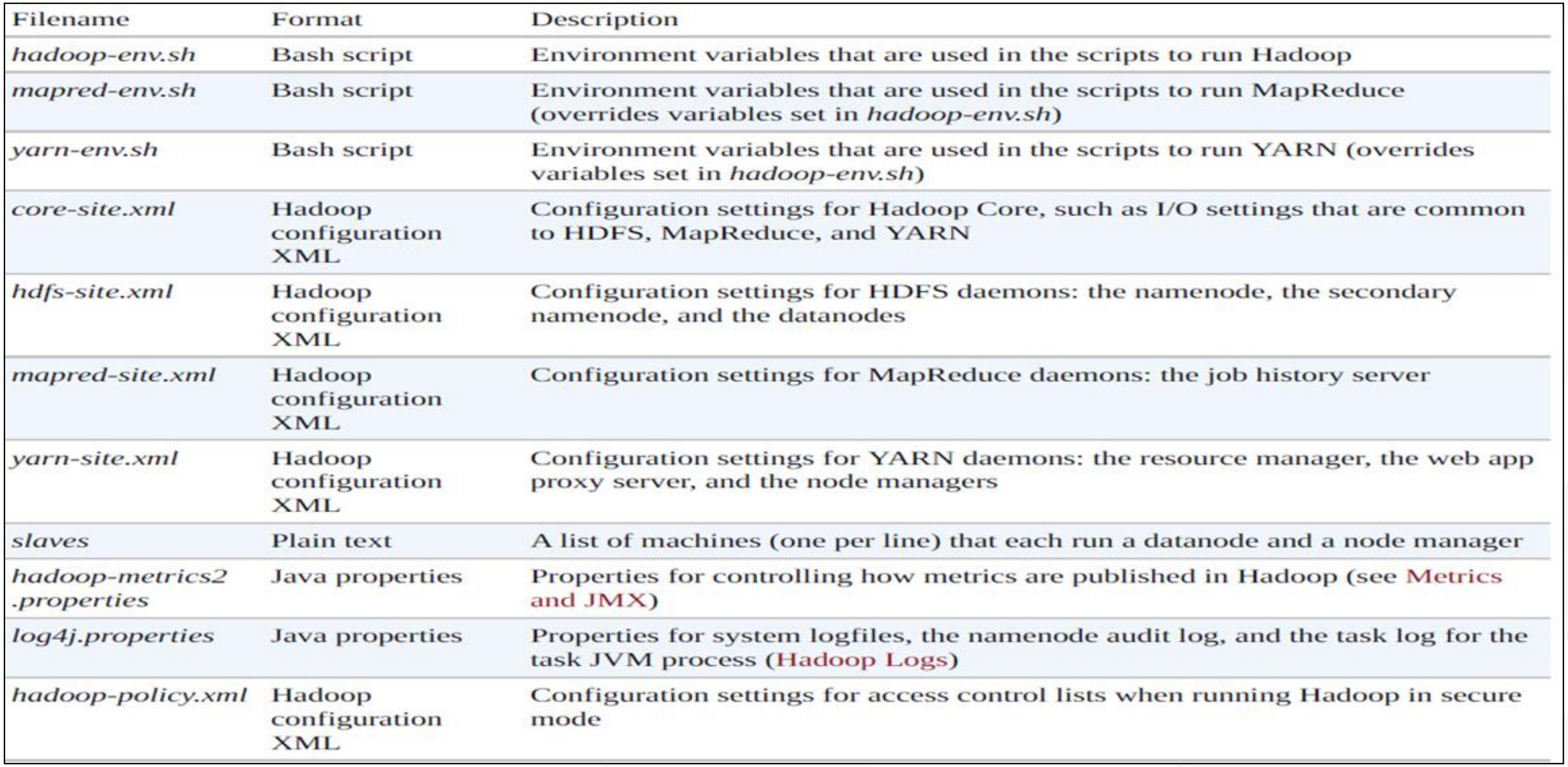
- QJM is dedicated HDFS implementation
- QJM runs as a group of Journal nodes and each edit must be written to majority of journal nodes
- QJM only allows one NameNode to write to the edit log at one time
- QJM has ssh fencing method implemented which helps to avoid the NameNode split-brain scenario

- ZooKeeper Failover Controller(ZKFC) is responsible for HA Monitoring for Name Node service and automatic failover
- There are two ZKFC process, one on each Name Node Machine
- ZKFC uses the ZooKeeper Service for coordination in determining which is the Active NameNode and in determining when to failover to the Standby NameNode.



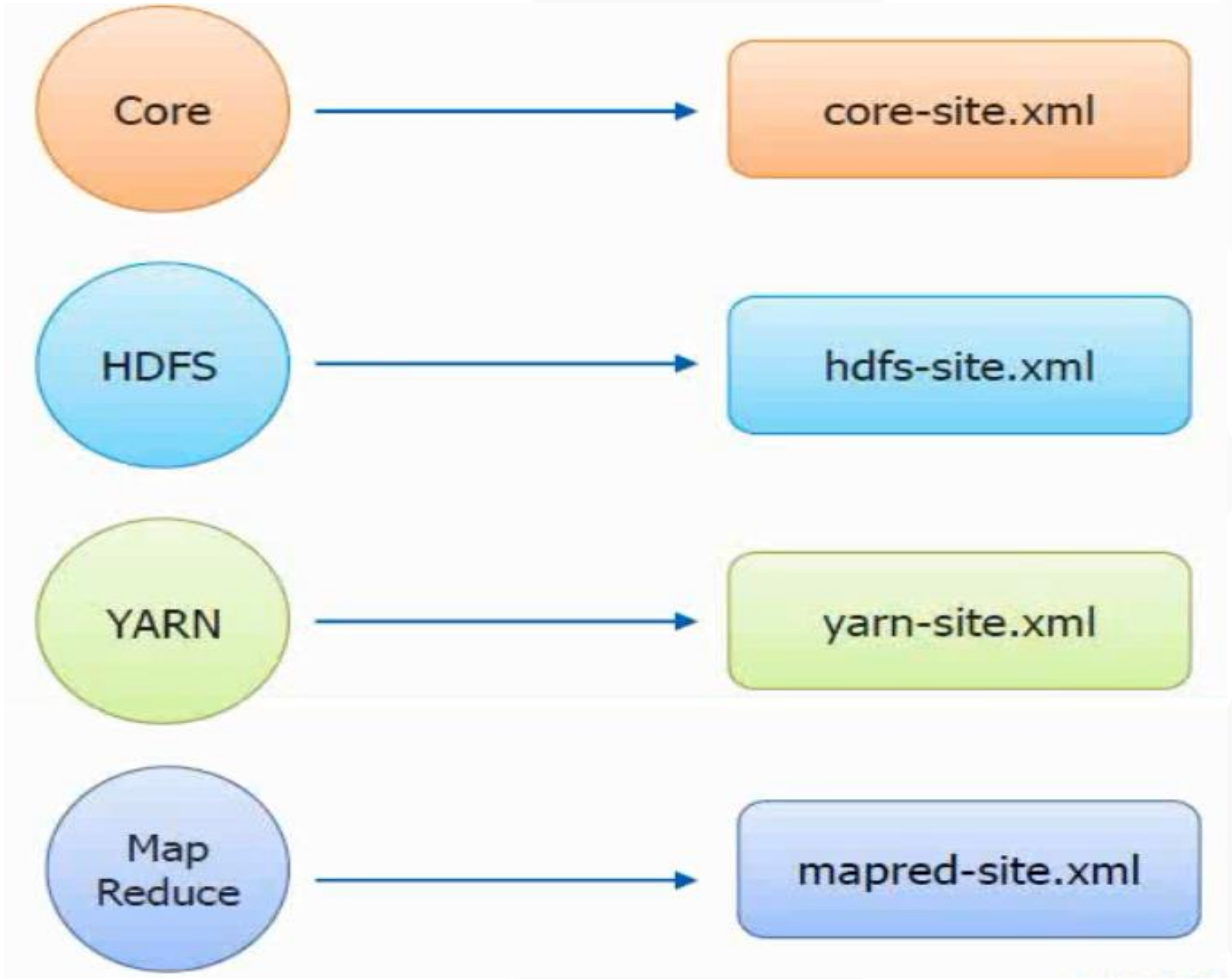
ANALYTIXLABS

Hadoop Configuration Files

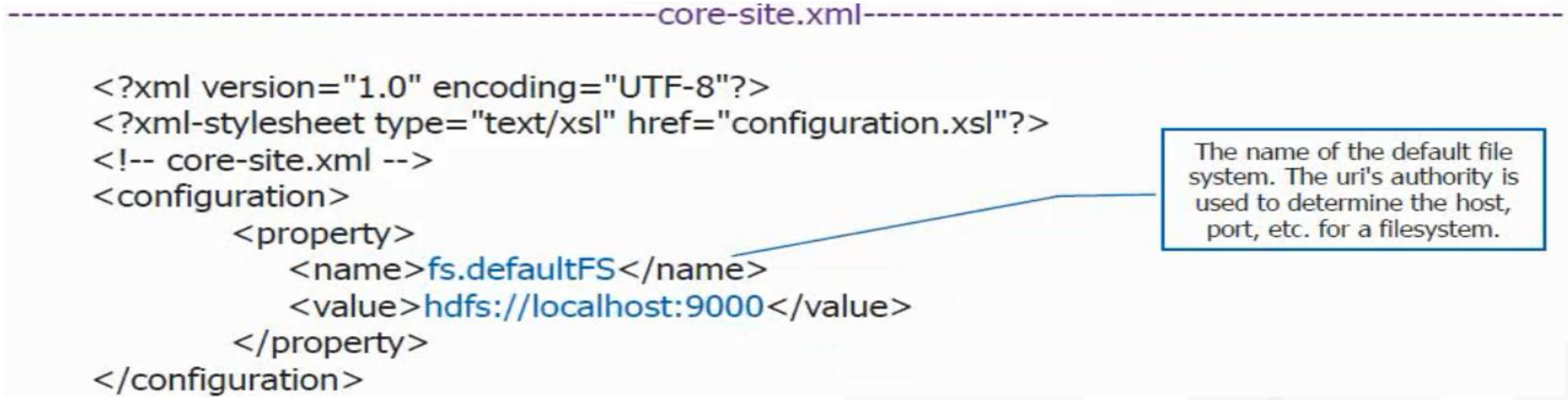


Filename	Format	Description
hadoop-env.sh	Bash script	Environment variables that are used in the scripts to run Hadoop
mapred-env.sh	Bash script	Environment variables that are used in the scripts to run MapReduce (overrides variables set in *hadoop-env.sh*)
yarn-env.sh	Bash script	Environment variables that are used in the scripts to run YARN (overrides variables set in *hadoop-env.sh*)
core-site.xml	Hadoop configuration XML	Configuration settings for Hadoop Core, such as I/O settings that are common to HDFS, MapReduce, and YARN
hdfs-site.xml	Hadoop configuration XML	Configuration settings for HDFS daemons: the namenode, the secondary namenode, and the datanodes
mapred-site.xml	Hadoop configuration XML	Configuration settings for MapReduce daemons: the job history server
yarn-site.xml	Hadoop configuration XML	Configuration settings for YARN daemons: the resource manager, the web app proxy server, and the node managers
slaves	Plain text	A list of machines (one per line) that each run a datanode and a node manager
hadoop-metrics2 .properties	Java properties	Properties for controlling how metrics are published in Hadoop (see Metrics and JMX)
log4j.properties	Java properties	Properties for system logfiles, the namenode audit log, and the task log for the task JVM process (Hadoop Logs)
hadoop-policy.xml	Hadoop configuration XML	Configuration settings for access control lists when running Hadoop in secure mode

Hadoop Core Configuration files



Hadoop Core Configuration files



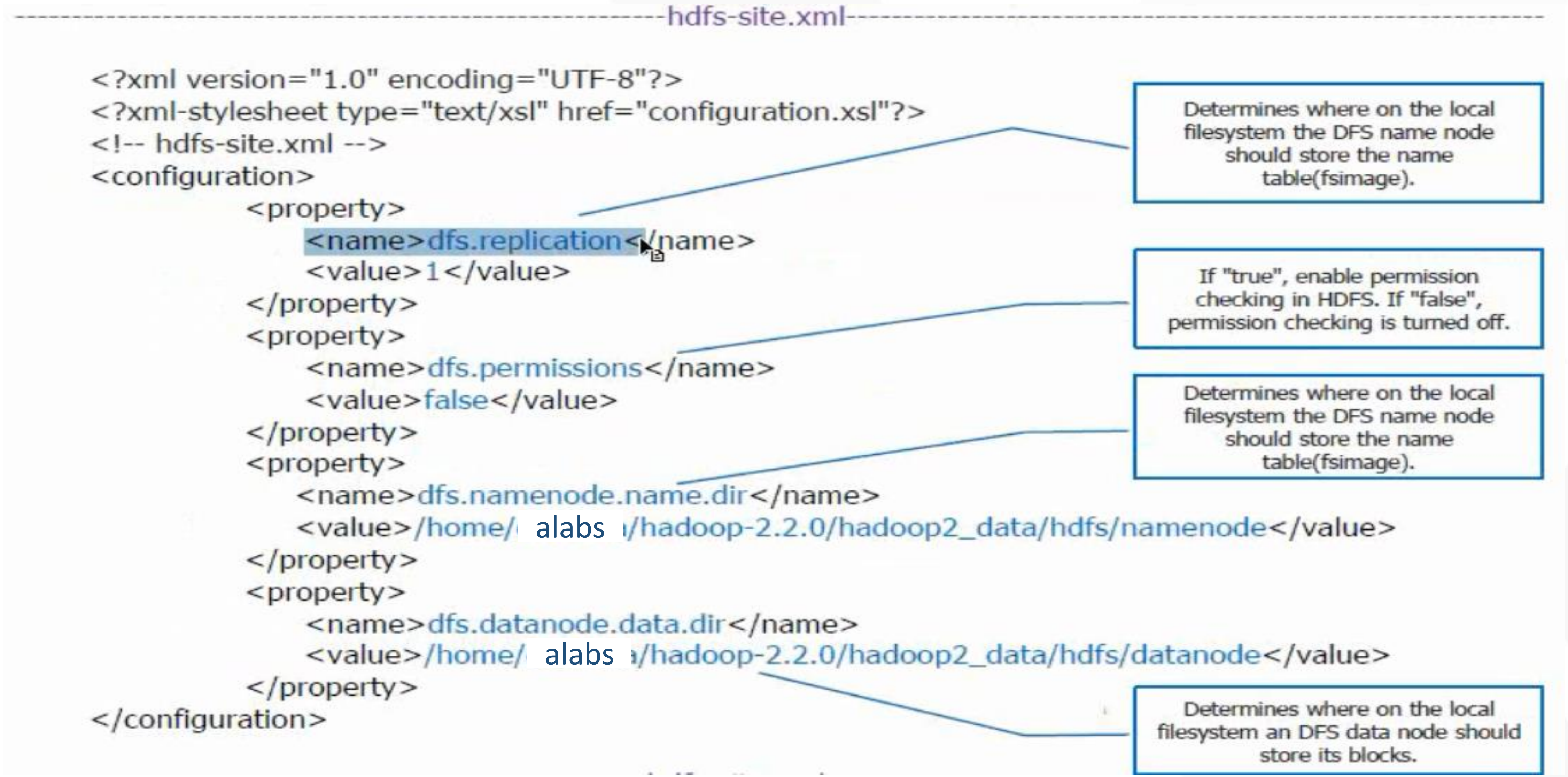
-----------------------------------core-site.xml-----------------------------------

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- core-site.xml -->
<configuration>
        <property>
            <name>fs.defaultFS</name>
            <value>hdfs://localhost:9000</value>
        </property>
</configuration>
```

The name of the default file system. The uri's authority is used to determine the host, port, etc. for a filesystem.

ANALYTIXLABS

Hadoop Core Configuration files



hdfs-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- hdfs-site.xml -->
<configuration>
        <property>
            <name>dfs.replication</name>
            <value>1</value>
        </property>
        <property>
            <name>dfs.permissions</name>
            <value>false</value>
        </property>
        <property>
           <name>dfs.namenode.name.dir</name>
           <value>/home/ alabs /hadoop-2.2.0/hadoop2_data/hdfs/namenode</value>
        </property>
        <property>
            <name>dfs.datanode.data.dir</name>
            <value>/home/ alabs /hadoop-2.2.0/hadoop2_data/hdfs/datanode</value>
        </property>
</configuration>
```

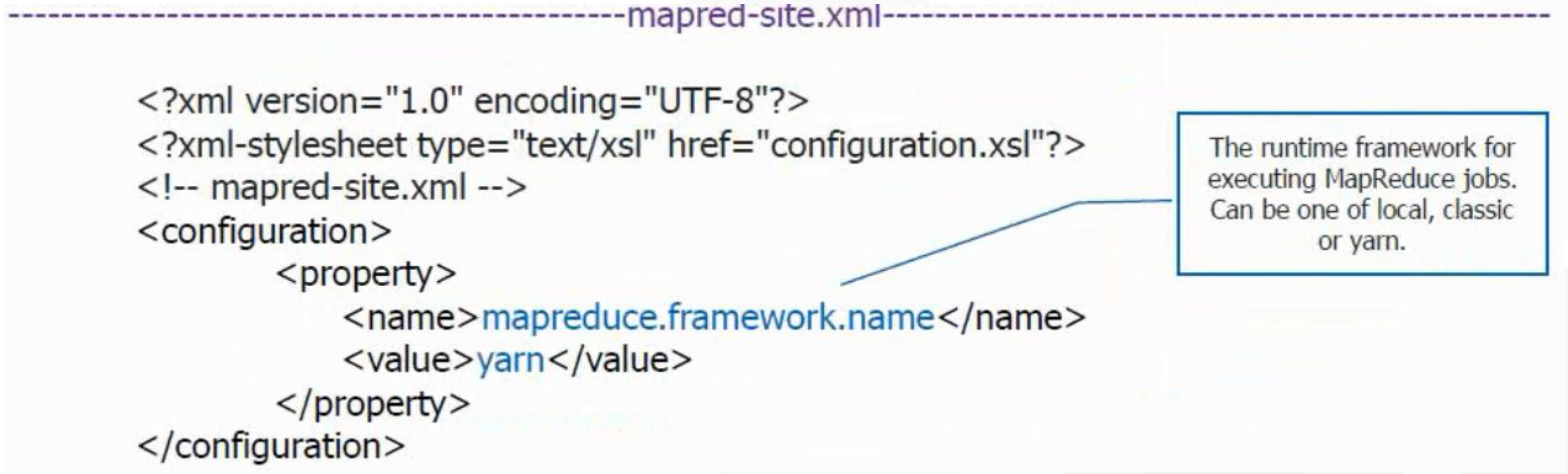
Determines where on the local filesystem the DFS name node should store the name table(fsimage).

If "true", enable permission checking in HDFS. If "false", permission checking is turned off.

Determines where on the local filesystem the DFS name node should store the name table(fsimage).

Determines where on the local filesystem an DFS data node should store its blocks.

Hadoop Core Configuration files

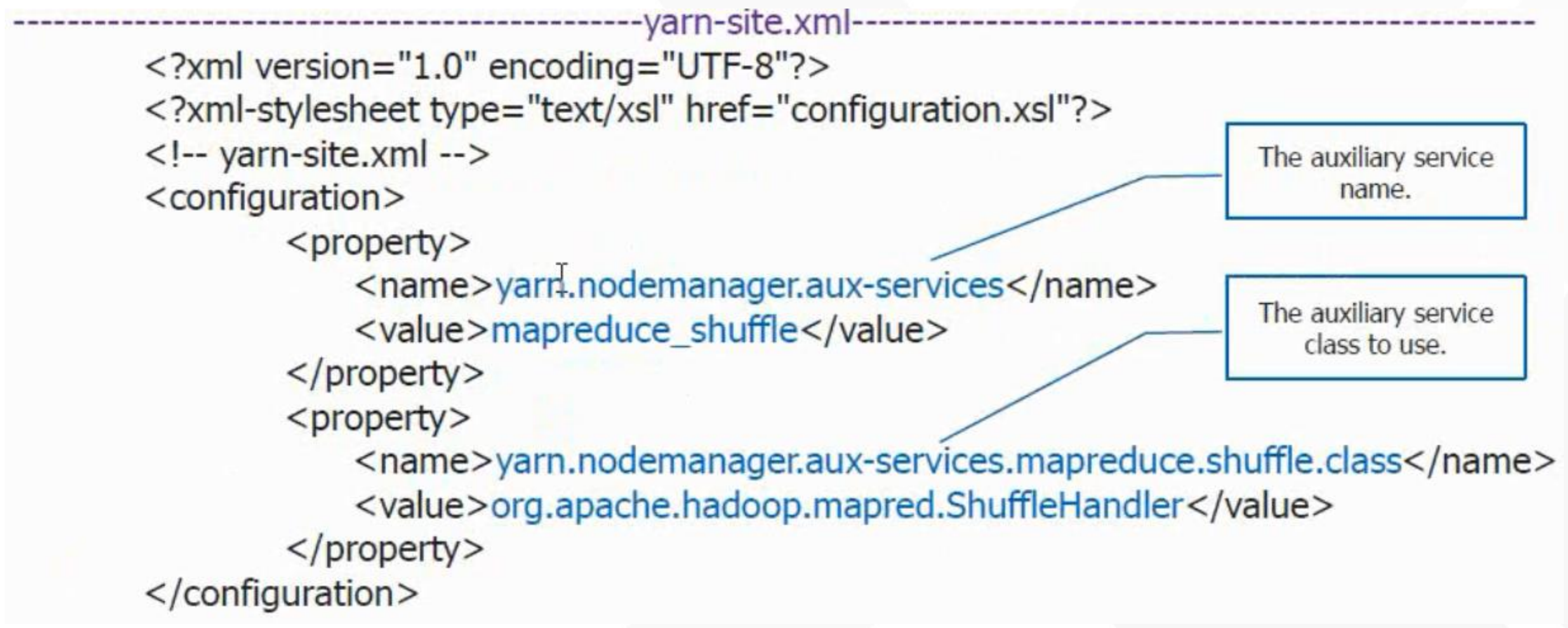


--mapred-site.xml--

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

The runtime framework for executing MapReduce jobs. Can be one of local, classic or yarn.

Hadoop Core Configuration files



```
-----------------------------------------yarn-site.xml-----------------------------------------
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- yarn-site.xml -->
<configuration>
        <property>
            <name>yarn.nodemanager.aux-services</name>
            <value>mapreduce_shuffle</value>
        </property>
        <property>
            <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
            <value>org.apache.hadoop.mapred.ShuffleHandler</value>
        </property>
</configuration>
```

The auxiliary service name.
The auxiliary service class to use.

Accessing HDFS

HDFS Access

HDFS provides following access mechanisms:

- FS Shell (Command Line)
- Web GUI (Hue)
- HTTP browser
- Java API for application

File System Commands: Linux - hdfs

Operation	UNIX/LINUX file system	HDFS File System
• To create a directory	mkdir <dirname>	hadoop fs -mkdir <dirname>
• To list the Contents of a directory/folder	ls –l <dirname>	hadoop fs -ls <dirname>
• To create an empty file	touch <filename>	hadoop fs -touchz <filename>
• To display the contents of a file terminal	cat <filename>	hadoop fs -cat <filename>
• To edit the contents of a file in editor	gedit <filename>	Not possible
• Copy a file from one location to another	cp <src> <dest>	hadoop fs -cp <src> <dest>
• Move a file from one location to another	mv <src> <dest>	hadoop fs -mv <src> <dest>
• Delete a file	rm <filename>	hadoop fs -rm <filename>
• Delete a directory	rm -r <dirname>	hadoop fs -rmr <dirname>
• **To copy a file from the local file system to HDFS**		**hadoop fs –put <src> <dest>**
• **To copy a file from HDFS to local file system**		**hadoop fs –get <src> <dest>**
• copy the files present in HDFS(path) to a single file in the local filesystem		**hadoop fs -getmerge /user/files merged.txt**

HDFS Command: Preparing HDFS filesystem

- Before it can be used, a brand new HDFS installation needs to be formatted
- Formatting process creates an empty filesystem by creating the storage directories and the initial version of namenode’s persistent data structure

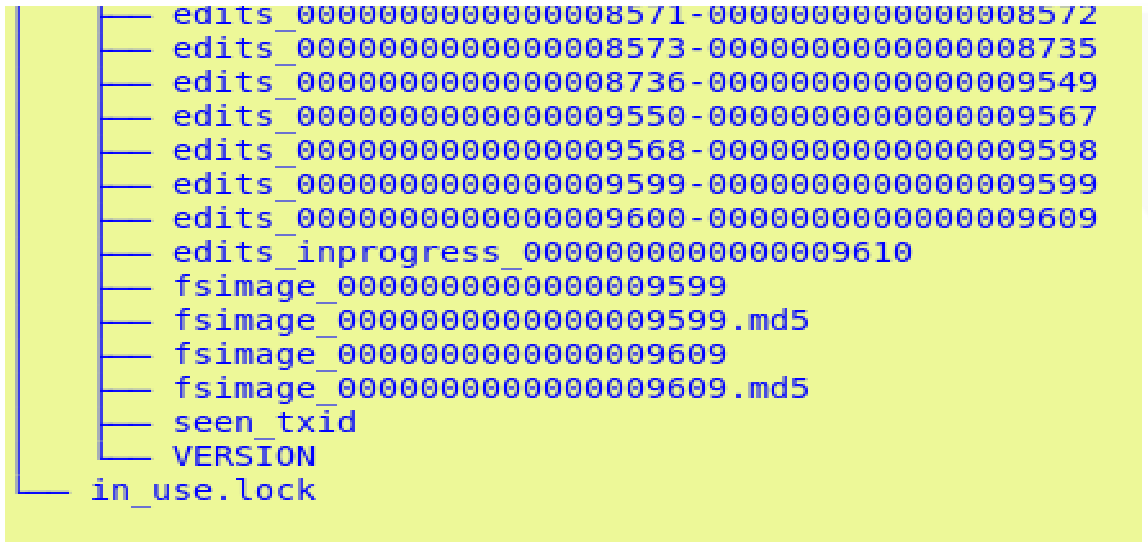
```
$ hdfs namenode –format
```

- Unlike namenodes, datanodes do not need to be explicitly formatted, because they create their storage directory automatically on startup
- Start HDFS demon

```
$ cd $HADOOP_HOME/sbin
$./start.dfs.sh
```

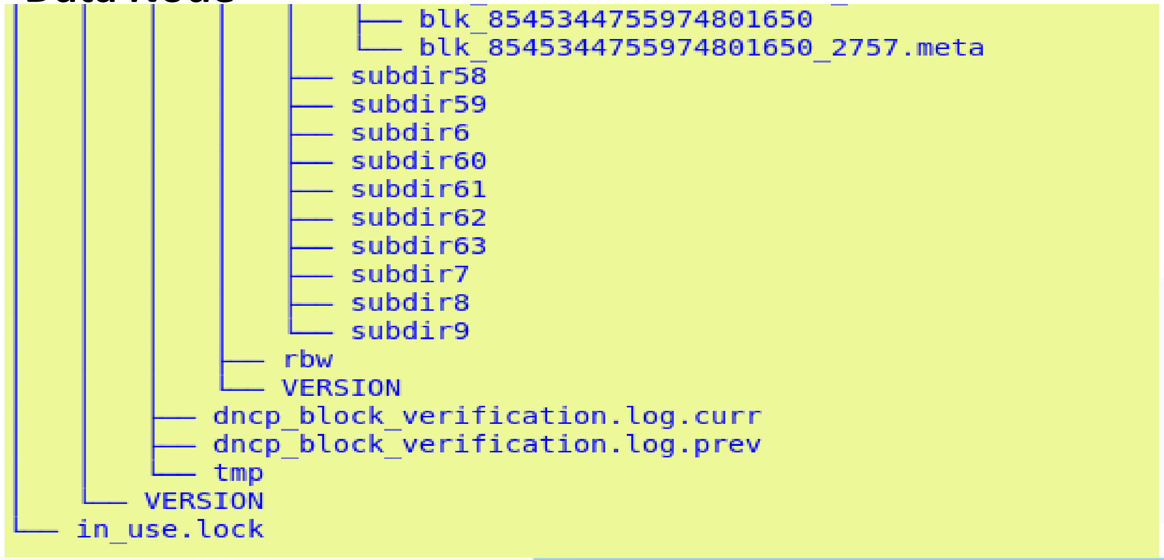
Namenode/Datanode directory structure

Name Node



```
│         ├── edits_0000000000000008571-0000000000000008572
│         ├── edits_0000000000000008573-0000000000000008735
│         ├── edits_0000000000000008736-0000000000000009549
│         ├── edits_0000000000000009550-0000000000000009567
│         ├── edits_0000000000000009568-0000000000000009598
│         ├── edits_0000000000000009599-0000000000000009599
│         ├── edits_0000000000000009600-0000000000000009609
│         ├── edits_inprogress_0000000000000009610
│         ├── fsimage_0000000000000009599
│         ├── fsimage_0000000000000009599.md5
│         ├── fsimage_0000000000000009609
│         ├── fsimage_0000000000000009609.md5
│         ├── seen_txid
│         └── VERSION
└── in_use.lock
```

Data Node



```
│   │   │   │   │   ├── blk_8545344755974801650
│   │   │   │   │   └── blk_8545344755974801650_2757.meta
│   │   │   │   ├── subdir58
│   │   │   │   ├── subdir59
│   │   │   │   ├── subdir6
│   │   │   │   ├── subdir60
│   │   │   │   ├── subdir61
│   │   │   │   ├── subdir62
│   │   │   │   ├── subdir63
│   │   │   │   ├── subdir7
│   │   │   │   ├── subdir8
│   │   │   │   └── subdir9
│   │   │   ├── rbw
│   │   │   └── VERSION
│   │   ├── dncp_block_verification.log.curr
│   │   ├── dncp_block_verification.log.prev
│   │   └── tmp
│   └── VERSION
└── in_use.lock
```

Hadoop HDFS Commands

1. Put Command
The ‘put’command feeds the data in to the HDFS.
Syntax: **hadoop dfs –put </source path> </destination path>**

2. List Command
The ‘list’command displays all the available files inside a particular path.
Syntax: **hadoop dfs –ls </source path>**

3.Get Command
The ‘get’ command copies the entire contents of the mentioned file to the local drive.
Syntax: **hadoop dfs –get </source path> </destination path>**

4. Make Directory Command
The ‘mkdir’ command creates a new directory in the specified location.
Syntax: **hadoop dfs –mkdir </source path>**

5. View contents of particular file
The ‘cat’ command is used to display all the contents of a file.
Syntax: **hadoop dfs –cat </path[filename]>**

Hadoop HDFS Commands

6.Duplicating a Complete File inside the HDFS.
The ‘copyfromlocal’ command will copy file from the local file system to the HDFS.
Syntax: **hadoop dfs –copyFromLocal </source path> </destination path>**

7.Duplicating a File from HDFS to the Local File System.
The ‘copytolocal’ command will copy files from the HDFS to the local file system.
Syntax: **hadoop dfs –copyToLocal </source path> </destination path>**

8.Removing the File
The command ‘rm’ will delete the file stored inside the HDFS.
Syntax: **hadoop dfs –rm </path[filename]>**

9.Run a DFS Filesystem to Check Utility
The command ‘fsck’ is used for checking the consistency of a file system
Syntax: hadoop fsck </file path>

10.Run a Cluster Balancing Utility
The command ‘balancer’ will check for work load on nodes in cluster and balance it.
Syntax: **hadoop balancer**

Hadoop HDFS Commands

11.Check Directory Space in HDFS
The command will show the file size occupied by file inside cluster.
Syntax: **hadoop dfs -du -s -h </file path>**

12. List all the Hadoop File System Shell Commands
The ‘fs’ command lists down all the shell commands of the Hadoop File System.
Syntax: **hadoop fs [options]**

13.Asking for Help
The ‘help’ command is for asking for help or querying a particular question.
Command: **hadoop fs -help**

Hadoop Admin Commands

1. hadoop fsck /
fsck command is used to check the HDFS file system. There are different arguments that can be passed with this command to emit different results.

2. hadoop fsck / -files
It displays all the files in HDFS while checking.

3. hadoop fsck / -files -blocks
It displays all the blocks of the files while checking.

4. hadoop fsck / -files –blocks -locations
It displays all the files block locations while checking.

5. **hadoop fsck / -files -blocks -locations -racks**
This command is used to display the networking topology for data-node locations.

6. hadoop fsck -delete
This command will delete the corrupted files in HDFS.

7. **hadoop fsck -move**
This command is used to move the corrupted files to a particular directory, by default it will move to the /lost+found directory.

Hadoop dfs Admin Commands

1. **hadoop dfsadmin -report**
This command is used to return the file system information and its statistics.

2.hadoop dfsadmin -metasave file_name.txt
This command is used to save the meta data that is present in the namenode in a file in the HDFS.

Here we have given the file_name as nn.txt and it has been saved in the log directory of hdfs.
3.**hadoop dfsadmin -refreshNodes**
This command is used to refresh the data nodes that are allowed to connect to the name node.

4.**hadoop fs -count -q /mydir**
Checks for the quota space for the specified directory or a file.

5.hadoop dfsadmin -setSpaceQuota 10M /dir_name
This command is used to set the space quota space for a particular directory. Now we will set the directory quota to 10MB and then we will check it using the command **hadoop fs -count -q /mydir.**

6.hadoop dfsadmin -clrSpaceQuota /mydir
This command is used to clear the allocated quota to a particular directory in HDFS. Now we will clear the quota which we have previously created and check the quota again.

Hadoop Safe Mode (Maintenance Mode) Commands

Hadoop Safe Mode (Maintenance Mode) Commands

Here we will discuss about Safe Mode commands:

1.hadoop dfsadmin -safemode enter

This command is used to enter safe mode.

2.hadoop dfsadmin -safemode leave

This command is used to leave safe mode.

3.hadoop dfsadmin -safemode get

This command is used to get the status of the safe mode.

Hadoop transparent Encryption(Security on data at rest)

```
# echo "sensetive information here" > test.file
# hadoop fs -put test.file /tmp/test.file
```

Intruder knows the file name and wants to get the content of it (sensitive information).

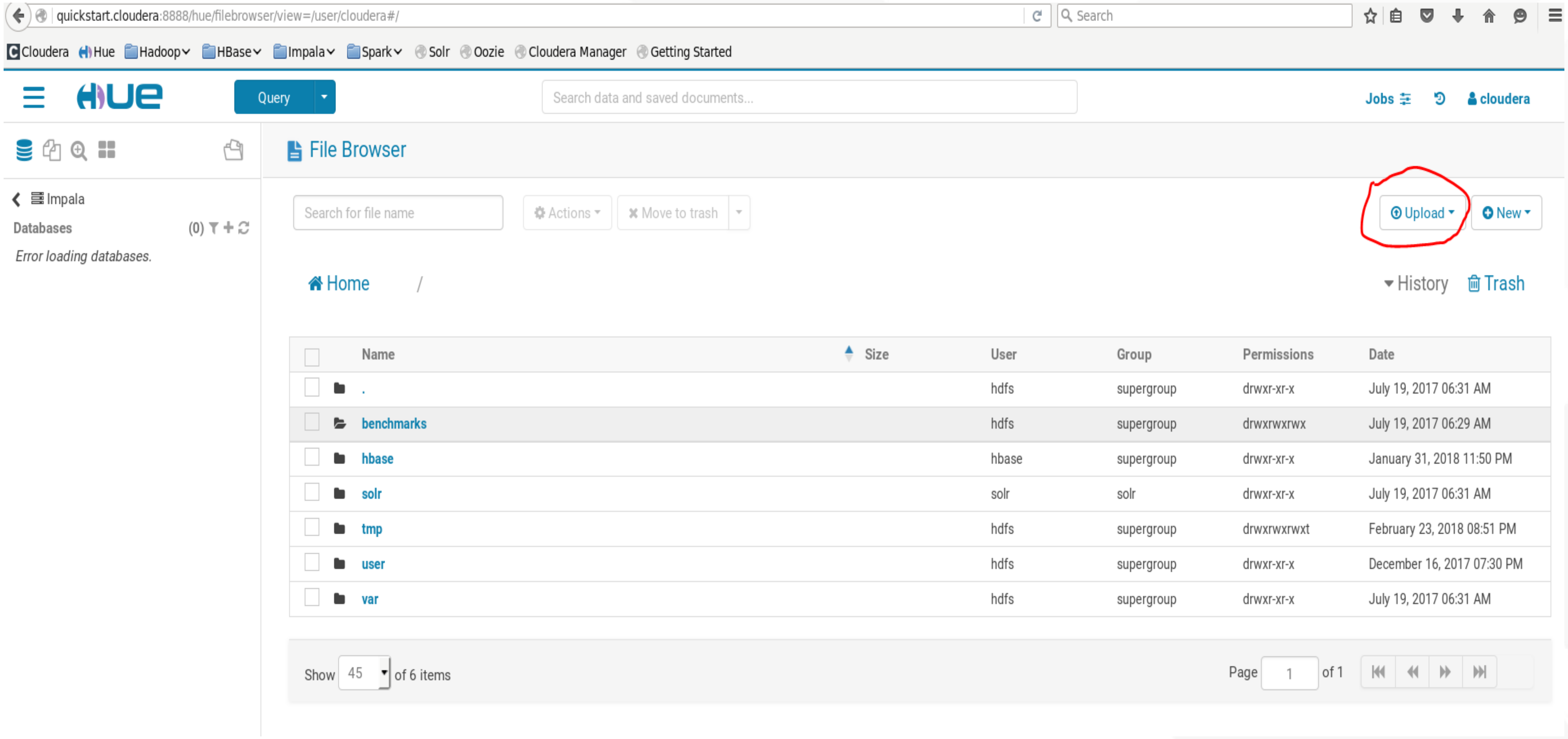
```
# hdfs fsck /tmp/test.file -locations -files -blocks
…
0. BP-421546782-192.168.254.66-1501133071977:blk_1073747034_6210 len=27 Live_repl=3 ...
# find / -name "blk_1073747034*"
/u02/hadoop/dfs/current/BP-421546782-192.168.254.66-1501133071977/current/finalized/subdir0/subdir20/blk_1073747034_6210.meta
/u02/hadoop/dfs/current/BP-421546782-192.168.254.66-1501133071977/current/finalized/subdir0/subdir20/blk_1073747034
# cat /u02/hadoop/dfs/current/BP-421546782-192.168.254.66-1501133071977/current/finalized/subdir0/subdir20/blk_1073747034

sensetive information here
------------------------------
// obtain hdfs kerberos ticket for working with cluster on behalf of hdfs user
# kinit -kt `find /var -name hdfs*keytab -printf "%T+\t%p\n" | sort|tail -1|cut -f 2` hdfs/`hostname`
// create directory, which will be our security zone in future
# hadoop fs -mkdir /tmp/EZ/
// create key in Key Trustee Server
# hadoop key create myKey
// create encrypted zone, using key created earlier
```

Hadoop transparent Encryption(Security on data at rest)

```
# hdfs crypto -createZone -keyName myKey -path /tmp/EZ
// make alabsuser owner of this directory
# hadoop fs -chown alabsuser:alabs /tmp/EZ
// Switch user to  alabsuser
# kinit -kt alabs.com.keytab alabsuser/alabs.com
// Load file in encrypted zone
# hadoop fs -put test.file /tmp/EZ/
// define physical location of the file on the Linus FS
# hadoop fsck /tmp/EZ/test.file -blocks -files -locations
..
0. BP-421546782-192.168.254.66-1501133071977:blk_1073747527_6703
...
[root@scajvm1bda01 ~]# find / -name blk_1073747527*
/u02/hadoop/dfs/current/BP-421546782-192.168.254.66-1501133071977/current/finalized/subdir0/subdir22/blk_1073747527_6703.meta
/u02/hadoop/dfs/current/BP-421546782-192.168.254.66-1501133071977/current/finalized/subdir0/subdir22/blk_1073747527
// trying to read the file
[root@scajvm1bda01 ~]# cat /u02/hadoop/dfs/current/BP-421546782-192.168.254.66-1501133071977/current/finalized/subdir0/subdir22/blk_1073747527
▒i#▒▒C▒x▒1▒U▒l▒▒▒[
```

Accessing HDFS using GUI(HUE)



Name	Size	User	Group	Permissions	Date
.		hdfs	supergroup	drwxr-xr-x	July 19, 2017 06:31 AM
benchmarks		hdfs	supergroup	drwxrwxrwx	July 19, 2017 06:29 AM
hbase		hbase	supergroup	drwxr-xr-x	January 31, 2018 11:50 PM
solr		solr	solr	drwxr-xr-x	July 19, 2017 06:31 AM
tmp		hdfs	supergroup	drwxrwxrwxt	February 23, 2018 08:51 PM
user		hdfs	supergroup	drwxr-xr-x	December 16, 2017 07:30 PM
var		hdfs	supergroup	drwxr-xr-x	July 19, 2017 06:31 AM

Show 45 of 6 items Page 1 of 1

HDFS Access using Rest API(webhdfs & HttpFS)

There are 2 different ways of accessing HDFS over http.

- **Using WebHDFS**

http://<active-namenode-server>:<namenode-port>/webhdfs/v1/<file-path>?op=OPEN

File Status: curl -i "http://localhost:50070/webhdfs/v1/tmp?user.name=cloudera&op=GETFILESTATUS"

Make directory: curl -i -X PUT "http://localhost:50070/webhdfs/v1/tmp/webhdfs?user.name=cloudera&op=MKDIRS"

Upload the file:

- curl -i -X PUT "http://localhost:50070/webhdfs/v1/tmp/webhdfs/webhdfs-test.txt?user.name=cloudera&op=CREATE"

- curl -i -T webhdfs-test.txt "http://localhost:50075/webhdfs/v1/tmp/webhdfs/webhdfs-test.txt?op=CREATE&user.name=cloudera&overwrite=false"

- **Using HttpFs(By Default its not provided by the cloudera. You have to install and it can act as proxy Server to access the hdfs behind the firewall**

http://<hadoop-httpfs-server>:<httpfs-port>/webhdfs/v1/<file-path>?op=OPEN

Accessing HDFS using Java Programming

There are two important classes in Hadoop:

To access the file system i.e. to create, read, write, delete etc. ---- FileSystem

To access the metadata of the files i.e. to find owner, block size etc.-- FileStatus

Import Java API's

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FSDataInputStream;
```

Program for List the files in a Directory

```
//list the contents of a file
 public class firstfile {
            public static void listFiles(FileSystem fs) throws IOException{
            Path path = new Path("/user/cloudera/");
            FileStatus[] status = fs.listStatus(path);
            for (FileStatus st : status) {
            System.out.println("Path : " + st.getPath().toString());
            System.out.println("Owner : " + st.getOwner().toString());
            System.out.println("is File? : " + st.isFile());
            }
}
```

Accessing HDFS using Java Programming

Program for Create and Write a file

```
public static void writeFile(FileSystem fs) throws IOException{
        FSDataOutputStream out;
        Path outFile = new Path("/user/cloudera/writeHdfs.txt");
        // Checks
        if (fs.exists(outFile)) {
            System.out.println("Output File Already exist");
            fs.delete(outFile);
        }
        out = fs.create(outFile);
        String input = "New File Created Today";
        out.write(input.getBytes());
        out.close();
}
```

Program for read data from file

```
public static void readFile(FileSystem fs) throws IOException {
Path inFile = new Path("/user/cloudera/writeHdfs.txt");
if (!fs.exists(inFile)) {
            System.out.println("Input file not found");
            System.exit(0); }
if (!fs.isFile(inFile)) {
            System.out.println("Input should be a file");
            System.exit(0); }
FSDataInputStream in = fs.open(inFile);
 BufferedReader br = new BufferedReader(new InputStreamReader(in));
String line = line = br.readLine();
while (line != null) { System.out.println(line);
    line = br.readLine();  }
in.close();        }
```

Accessing HDFS using Java Programming

Main method

```
public static void main(String[] args) throws IOException {
    //Configuration Object
    Configuration conf = new Configuration();
    conf.addResource(new Path("/usr/lib/hadoop/etc/hadoop/core-site.xml"));
    conf.addResource(new Path("/usr/lib/hadoop/etc/hadoop/hdfs-site.xml"));
    // FileSystem object
    FileSystem fs = FileSystem.get(conf);
    listFiles(fs);
    writeFile(fs);
    readFile(fs);
     fs.close();
  }
}
```

Contact Us

Visit us on: http://www.analytixlabs.in/

For more information, please contact us: http://www.analytixlabs.co.in/contact-us/

Or email: info@analytixlabs.co.in

Call us we would love to speak with you: (+91) 9910509849

Join us on:

Twitter - http://twitter.com/#!/AnalytixLabs

Facebook - http://www.facebook.com/analytixlabs

LinkedIn - http://www.linkedin.com/in/analytixlabs

Blog - http://www.analytixlabs.co.in/category/blog/