Hadoop admin day to day activities:

***Day to day responsibilities of a Hadoop Administrator.***

* Installing Hadoop in Linux environment.
* Tuning of Hadoop cluster and Hadoop MapReduce routines.
* Backup and recovery tasks.
* Configuring NameNode to ensure its high availability.
* Deployment in Hadoop cluster and maintaining it.
* Health check of Hadoop cluster monitoring.
* Support and maintenance of HDFS.
* Monitor Hadoop cluster connectivity and security.
* Manage Hadoop log files.
* Resource and security management.
* Troubleshoot application errors.
* Propose and deploy new hardware and software environment required for Hadoop.

1. Installation of the cluster through parcels using Ansible.
2. Adding/removing nodes and services to the cloudera manager and validation of services.
3. Adding and deleting user spaces and setting up quotas.
4. Changing cluster capacity scaling up and scaling out.
5. Adding/removing user to cloudera and giving appropriate permissions.
6. Configuring alerts and notifications.
7. Supporting other teams like developers and DA to run jobs

Que:How will you increase hdfs balance performance , if hdfs balancer is running slow?

# Ans: HDFS Balancers

HDFS data might not always be distributed uniformly across DataNodes. One common reason is addition of new DataNodes to an existing cluster. HDFS provides a balancer utility that analyzes block placement and balances data across the DataNodes. The balancer moves blocks until the cluster is deemed to be balanced, which means that the utilization of every DataNode (ratio of used space on the node to total capacity of the node) differs from the utilization of the cluster (ratio of used space on the cluster to total capacity of the cluster) by no more than a given threshold percentage. The balancer does not balance between individual volumes on a single DataNode.

Continue reading:

* [Configuring and Running the HDFS Balancer Using Cloudera Manager](http://bdlabs.edureka.co/static/help/topics/admin_hdfs_balancer.html#cmug_topic_5_14_unique_1)
* [Configuring and Running the HDFS Balancer Using the Command Line](http://bdlabs.edureka.co/static/help/topics/admin_hdfs_balancer.html#xd_583c10bfdbd326ba--6eed2fb8-14349d04bee--780a_unique_1)

## Configuring and Running the HDFS Balancer Using Cloudera Manager

**Minimum Required Role:** [**Cluster Administrator**](http://bdlabs.edureka.co/static/help/topics/cm_sg_user_roles.html#concept_wfh_tvy_qp_unique_1) (also provided by **Full Administrator**)

In Cloudera Manager, the HDFS balancer utility is implemented by the Balancer role. The Balancer role usually shows a health of **None** on the HDFS Instances tab because it does not run continuously.

The Balancer role is normally added (by default) when the HDFS service is installed. If it has not been added, you must add a Balancer role to rebalance HDFS and to see the **Rebalance** action.

### Configuring the Balancer Threshold

The Balancer has a default threshold of 10%, which ensures that disk usage on each DataNode differs from the overall usage in the cluster by no more than 10%. For example, if overall usage across all the DataNodes in the cluster is 40% of the cluster's total disk-storage capacity, the script ensures that DataNode disk usage is between 30% and 50% of the DataNode disk-storage capacity. To change the threshold:

1. Go to the HDFS service.
2. Click the **Configuration** tab.
3. Select **Scope** > **Balancer**.
4. Select **Category** > **Main**.
5. Set the **Rebalancing Threshold** property.

If more than one role group applies to this configuration, edit the value for the appropriate role group. See [Modifying Configuration Properties Using Cloudera Manager](http://bdlabs.edureka.co/static/help/topics/cm_mc_mod_configs.html#cmug_topic_5_3).

1. Click **Save Changes** to commit the changes.

### Configuring Concurrent Moves

The property dfs.datanode.balance.max.concurrent.moves sets the maximum number of threads used by the DataNode balancer for pending moves. It is a throttling mechanism to prevent the balancer from taking too many resources from the DataNode and interfering with normal cluster operations. Increasing the value allows the balancing process to complete more quickly, decreasing the value allows rebalancing to complete more slowly, but is less likely to compete for resources with other tasks on the DataNode. To use this property, you need to set the value on both the DataNode and the Balancer.

* To configure the Datanode:
  + Go to the HDFS service.
  + Click the **Configuration** tab.
  + Search for **DataNode Advanced Configuration Snippet (Safety Valve) for hdfs-site.xml**.
  + Add the following code to the configuration field, for example, setting the value to 50.
  + <property>
  + <name>dfs.datanode.balance.max.concurrent.moves</name>
  + <value>50</value>

</property>

* + Restart the DataNode.
* To configure the Balancer:
  + Go to the HDFS service.
  + Click the **Configuration** tab.
  + Search for **Balancer Advanced Configuration Snippet (Safety Valve) for hdfs-site.xml**.
  + Add the following code to the configuration field, for example, setting the value to 50.
  + <property>
  + <name>dfs.datanode.balance.max.concurrent.moves</name>
  + <value>50</value>

</property>

### Running the Balancer

1. Go to the HDFS service.
2. Ensure the service has a Balancer role.
3. Select **Actions** > **Rebalance**.
4. Click **Rebalance** to confirm. If you see a **Finished** status, the Balancer ran successfully.

## Configuring and Running the HDFS Balancer Using the Command Line

http://bdlabs.edureka.co/static/help/static/important.jpg  **Important**:

* Follow these command-line instructions on systems that do not use Cloudera Manager.
* This information applies specifically to CDH 5.11.x. See [Cloudera Documentation](http://www.cloudera.com/content/support/en/documentation.html) for information specific to other releases.

The HDFS balancer re-balances data across the DataNodes, moving blocks from overutilized to underutilized nodes. As the system administrator, you can run the balancer from the command-line as necessary -- for example, after adding new DataNodes to the cluster.

Points to note:

* The balancer requires the capabilities of an HDFS superuser (for example, the hdfs user) to run.
* The balancer does not balance between individual volumes on a single DataNode.
* You can run the balancer without parameters, as follows:

sudo -u hdfs hdfs balancer

http://bdlabs.edureka.co/static/help/static/note.jpg  **Note**: If [Kerberos is enabled](http://bdlabs.edureka.co/static/help/topics/cdh_sg_cdh5_hadoop_security.html#topic_3_unique_4), do not use commands in the form sudo -u <user> hadoop <command>; they will fail with a security error. Instead, use the following commands: $ kinit <user> (if you are using a password) or $ kinit -kt <keytab> <principal> (if you are using a keytab) and then, for each command executed by this user, $ <command>

This runs the balancer with a default threshold of 10%, meaning that the script will ensure that disk usage on each DataNode differs from the overall usage in the cluster by no more than 10%. For example, if overall usage across all the DataNodes in the cluster is 40% of the cluster's total disk-storage capacity, the script ensures that each DataNode's disk usage is between 30% and 50% of that DataNode's disk-storage capacity.

* You can run the script with a different threshold; for example:

sudo -u hdfs hdfs balancer -threshold 5

This specifies that each DataNode's disk usage must be (or will be adjusted to be) within 5% of the cluster's overall usage.

* You can adjust the network bandwidth used by the balancer, by running the dfsadmin -setBalancerBandwidth command before you run the balancer; for example:

dfsadmin -setBalancerBandwidth newbandwidth

where newbandwidth is the maximum amount of network bandwidth, in bytes per second, that each DataNode can use during the balancing operation. For more information about the setBalancerBandwidth and other HDFS command-line options, see the [dfsadmin](https://hadoop.apache.org/docs/r2.7.1/hadoop-project-dist/hadoop-hdfs/HDFSCommands.html#dfsadmin) documentation.

* The property dfs.datanode.balance.max.concurrent.moves sets the maximum number of threads used by the DataNode balancer for pending moves. It is a throttling mechanism to prevent the balancer from taking too many resources from the DataNode and interfering with normal cluster operations. Increasing the value allows the balancing process to complete more quickly, decreasing the value allows rebalancing to complete more slowly, but is less likely to compete for resources with other tasks on the DataNode. Adjust the value of this property in the /etc/hadoop/[service name]/hdfs-site.xml configuration file.
* <property>
* <name>dfs.datanode.balance.max.concurrent.moves</name>
* <value>50</value>

</property>

Q2:How to enable HA in Kerberos ?

Ans: We will implement new KDC and add new second KDC server config in krb5.conf.

## **Which are the modes in which Hadoop can run?**

We have three modes in which Hadoop can run and that are:

|  |  |
| --- | --- |
| **Modes to run Hadoop** | |
| **Modes** | **Description** |
| **Standalone (local)** | * It is the default mode of Hadoop that uses the local file system for input and output operations. * This mode is mainly used for debugging purpose, and it does not support the use of HDFS. * No custom configuration required for mapred-site.xml, core-site.xml, hdfs-site.xml files. |
| **Pseudo-distributed** | * You need configuration for all the three files mentioned above. * All daemons are running on one node. * Both Master & Slave nodes are on the same machine. |
| **Fully distributed** | * It is a production phase of Hadoop. * Data is distributed across several nodes on a Hadoop cluster. * Separate nodes are allotted as Master and Slaves. |

## **2. What are the features of Standalone (local) mode?**

* In stand-alone mode, there are no daemons, everything runs on a single JVM.
* It has no DFS and it utilizes the local file system.
* Stand-alone mode is suitable only for running MapReduce programs during development for testing.
* It is one of the least used environments.

## **3. What are the features of Pseudo mode?**

Pseudo mode is used in both for development and in the testing environment. In the Pseudo mode, all the daemons run on the same machine.

## **4. What are the features of Fully Distributed mode?**

This is an important question as Fully Distributed mode is used in the production environment, where we have ‘n’ number of machines forming a Hadoop cluster. Hadoop daemons run on a cluster of machines. There is one host onto which Namenode is running and other hosts on which Datanodes are running. NodeManagers are installed on every DataNode and it is responsible for execution of the task on every single DataNode. All these NodeManagers are managed by ResourceManager, which receives the processing requests, and then passes the parts of requests to corresponding NodeManagers accordingly.

## **5. What is configured in /etc/hosts and what is its role in setting Hadoop cluster?**

This is a technical question which challenges your basic concept. /etc/hosts file contains the hostname and their IP address of that host. It maps the IP address to the hostname. In Hadoop cluster, we store all the hostnames (master and slaves) with their IP addresses in /etc/hosts so, that we can use hostnames easily instead of IP addresses.

## **6. What are the default port numbers of NameNode, ResourceManager & MapReduce JobHistory Server?**

You are expected to remember basic server port numbers if you are working with Hadoop. The port number for corresponding daemons are as follows:

Namenode – ’50070’

ResourceManager – ’8088’

MapReduce JobHistory Server – ’19888’.

## **7. What are the main Hadoop configuration files?**

♣ Tip: Generally, approach this question by telling the 4 main configuration files in Hadoop and giving their brief descriptions to show your expertise.

* **core-site.xml:**core-site.xml informs Hadoop daemon where NameNode runs on the cluster. It contains configuration settings of Hadoop core such as I/O settings that are common to HDFS & MapReduce.
* **hdfs-site.xml:** 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.
* **mapred-site.xml**: mapred-site.xml contains configuration settings of the MapReduce framework like number of JVM that can run in parallel, the size of the mapper and the reducer, CPU cores available for a process, etc.
* **yarn-site.xml:**yarn-site.xml contains configuration settings of ResourceManager and NodeManager like application memory management size, the operation needed on program & algorithm, etc.

These files are in the conf/hadoop/ directory inside Hadoop directory.

## **8. How does Hadoop CLASSPATH plays vital role in starting or stopping in Hadoop daemons?**

♣ Tip: To check your knowledge on Hadoop the interviewer may ask you this question.

CLASSPATH includes all the directories containing jar files required to start/stop Hadoop daemons. The CLASSPATH is set inside **/etc/hadoop/hadoop-env.sh** file.

## **9. What is a spill factor with respect to the RAM?**

♣ Tip: This is a theoretical question, but if you add a practical taste to it, you might get a preference.

The map output is stored in an in-memory buffer; when this buffer is almost full, then spilling phase starts in order to move data to a temp folder.

Map output is first written to buffer and buffer size is decided by mapreduce.task.io.sort.mb property .By default, it will be 100 MB.

When the buffer reaches certain threshold, it will start spilling buffer data to disk. This threshold is specified in mapreduce.map.sort.spill.percent .

## **10. What is command to extract the compressed file in tar.gz format?**

This is an easy question,**tar -xvf /file\_location/filename.tar.gz**command will extract the tar.gz compressed file.

**11. How will you check Java and Hadoop is installed on your system?**

By using the following commands we can check whether Java and Hadoop are installed and their paths are set inside .bashrc file:

For checking Java –**java -version**

For checking Hadoop –**hadoop version**

## **12. What is the default replication factor and how will you change it?**

The default replication factor is 3.

♣ Tip: Default Replication Factor could be changed in three ways. Answering all the three ways will show your expertise.

1. By adding this property to **hdfs-site.xml**:

|  |  |
| --- | --- |
| 1  2  3  4  5 | <property>  <name>dfs.replication</name>  <value>5</value>  <description>Block Replication</description>  </property> |

1. Or you can change the replication factor on per file basis using following command:

**hadoop fs –setrep –w 3 /file\_location**

1. Or you can change replication factor for all the files in a directory using the following command:

**hadoop fs –setrep –w 3 -R /directory\_location**

## **13. What is the full form of fsck?**

The full form of fsck is File System Check. HDFS supports the fsck (filesystem check) command to check for various inconsistencies. It is designed for reporting the problems with the files in HDFS, for example, missing blocks of a file or under-replicated blocks.

## **14. Which are the main hdfs-site.xml properties?**

The three main hdfs-site.xml properties are:

1. **dfs.name.dir**gives you the location where NameNode stores the metadata (FsImage and edit logs) and where DFS is located – on the disk or onto the remote directory.
2. **dfs.data.dir**which gives you the location of DataNodes, where the data is going to be stored.
3. **fs.checkpoint.dir** is the directory on the filesystem where the Secondary NameNode stores the temporary images of edit logs, which is to be merged and the FsImage for backup

## **15. What happens if you get a ‘connection refused java exception’ when you type hadoop fsck /?**

If you get a ‘connection refused java exception’ when you type hadoop fsck, it could mean that the NameNode is not working on your VM.

## **16. How can we view the compressed files via HDFS command?**

We can view compressed files in HDFS using **hadoop fs -text /filename**command.

## **17. What is the command to move into safe mode and exit safe mode?**

♣ Tip: Approach this question by first explaining safe mode and then moving on to the commands.

Safe Mode in Hadoop is a maintenance state of the NameNode during which NameNode doesn’t allow any changes to the file system. During Safe Mode, HDFS cluster is read-only and doesn’t replicate or delete blocks.

* To know the status of safe mode, you can use the command: ***hdfs dfsadmin -safemode get***
* To enter safe mode: ***hdfs dfsadmin -safemode enter***
* To exit safe mode: ***hdfs dfsadmin -safemode leave***

## **18. What does ‘jps’ command does?**

**jps** command is used to check all the Hadoop daemons like NameNode, DataNode, ResourceManager, NodeManager etc. which are running on the machine.

## **19. How can I restart Namenode?**

This question has two answers, answering both will give you a plus point. We can restart NameNode by following methods:

1. You can stop the NameNode individually using**.** **/sbin /hadoop-daemon.sh stop namenode**command and then start the NameNode using**.** **/sbin/hadoop-daemon.sh start namenode**
2. Use .**/sbin/stop-all.sh**and and then use .**/sbin/start-all.sh** command which will stop all the daemons first and then start all the daemons.

## **20. How can we check whether NameNode is working or not?**

To check whether NameNode is working or not, use the **jps** command, this will show all the running Hadoop daemons and there you can check whether NameNode daemon is running or not**.**

## **21. How can we look at the Namenode in the web browser UI?**

If you want to look for NameNode in the browser, the port number for NameNode web browser UI is **50070**. We can check in web browser using http://master:50070/dfshealth.jsp.

## **22. What are the different commands used to startup and shutdown Hadoop daemons?**

♣ Tip: Explain all the three ways of stopping and starting Hadoop daemons, this will show your expertise.

1. **./sbin/start-all.sh** to start all the Hadoop daemons and .**/sbin/stop-all.sh** to stop all the Hadoop daemons.
2. Then you can start all the dfs daemons together using**.** **/sbin/start-dfs.sh**, yarn daemons together using**.** **/sbin/start-yarn.sh**and MR job history server using**.** **/sbin/mr-jobhistory-daemon.sh start historyserver**. To stop these daemons similarly we can use**.** **/sbin/start-yarn.sh**, **./sbin/start-yarn.sh**&**.** **/sbin/mr-jobhistory-daemon.sh stop historyserver**.
3. The last way is to start all the daemons individually and stop them individually:

**./sbin/hadoop-daemon.sh start namenode**

**./sbin/hadoop-daemon.sh start datanode**

**./sbin/yarn-daemon.sh start resourcemanager**

**./sbin/yarn-daemon.sh start nodemanager**

**./sbin/mr-jobhistory-daemon.sh start historyserver**

and stop them similarly.

## **23. What do slaves file consist of?**

Slaves file consists of a list of hosts, one per line and the list contains DataNode location on which Node Manager servers run.

## **24. What do masters file consist of?**

The masters file contains Secondary NameNode server location.

## **25. What does hadoop-env.sh do?**

**hadoop-env.sh** provides the environment for Hadoop to run. For example, JAVA\_HOME, CLASSPATH etc. are set over here.

## **26. Where is hadoop-env.sh file present?**

As we discussed earlier, where all the configuration files reside, thus hadoop-env.sh file is present in the  /etc/hadoop directory.

## **27. In Hadoop\_PID\_DIR, what does PID stands for? What does it do?**

PID stands for ‘Process ID’. This directory stores the Process ID of the servers that are running.

## **28. What does hadoop-metrics.properties file do?**

♣ Tip: As this file is configured manually only in special cases, so answering this question will impress the interviewer indicating your expertise about configuration files.

**hadoop-metrics.properties** is used for ‘Performance Reporting‘ purposes. It controls the reporting for Hadoop. The API is abstract so that it can be implemented on top of a variety of metrics client libraries. The choice of client library is a configuration option, and different modules within the same application can use different metrics implementation libraries. This file is stored inside **/etc/hadoop**.

## **29. What are the network requirements for Hadoop?**

You should answer this question as, the Hadoop core uses Shell (SSH) for communication with salve and to launch the server processes on the slave nodes. It requires a password-less SSH connection between the master and all the slaves and the secondary machines, so every time it does not have to ask for authentication as master and slave requires rigorous communication.

## **30. Why do we need a password-less SSH in Fully Distributed environment?**

We need a password-less SSH in a Fully-Distributed environment because when the cluster is live and running in Fully Distributed environment, the communication is too frequent. The DataNode and the NodeManager should be able to send messages quickly to master server.

## **31. Does this lead to security issues?**

No, not at all. Hadoop cluster is an isolated cluster and generally, it has nothing to do with the internet. It has a different kind of a configuration. We needn’t worry about that kind of a security breach, for instance, someone hacking through the internet, and so on. Hadoop has a very secured way to connect to other machines to fetch and to process data.

## **32. On which port does SSH work?**

SSH works on Port No. **22,**though it can be configured. **22** is the default Port number.

## **33. Can you tell us more about SSH?**

SSH is nothing but a secure shell communication, it is a kind of a protocol that works on a Port No. 22, and when you do an SSH, what you really require is a password, to connect to the other machine. SSH is not only between masters and slaves, but can be between two hosts.

## **34. What happens to a NameNode, when ResourceManager is down?**

When a ResourceManager is down, it will not be functional (for submitting jobs) but NameNode will be present. So, the cluster is accessible if NameNode is working, even if the ResourceManager is not working.

## 35. How can we format HDFS?

♣ Tip: Attempt this question by starting with the command to format the HDFS and then exlain what this command does.

Hadoop distributed file system(HDFS) can be formatted using ***bin/hadoop namenode -format***command. This command 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. If you run this command on existing filesystem, you will lose all your data stored on your NameNode. Formatting a Namenode will not format the DataNode. It will format the FsImage and edit logs data stored on the NameNode and will lose the data about the location of blocks stored in HDFS.

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

Hdfs questions and answers:

Q: How does a NameNode handle the failure of the DataNodes in Hadoop?.

Ans: HDFS has master-slave architecture in which master is namenode and slave is datanode.  
HDFS cluster has single namenode that manages file system namespace (metadata) and multiple datanodes that are responsible for storing actual data in HDFS and performing the read-write operation as per request for the clients.

In HDFS, each DataNode sends Heartbeat and Data Blocks report to NameNode. Receipt of a heartbeat implies that the datanode is functioning properly. A block report contains a list of all blocks on a datanode.  
Data node passes a heartbeat signal to Name node in an interval of 2 minutes.When Name node does not receive heartbeat signals from Data node, it assumes that the data node is either dead or non-functional.  
As soon as the data node is declared dead/non-functional all the data blocks it hosts are transferred to the other data nodes with which the blocks are replicated initially. This is how Namenode handles datanode failures.

**Q:Why check-pointing process need to be done by only secondary namenode/ checkpoint node?. Why not namenode?**

Ans :For suppose namenode is also taking care of check-pointing process. To perform check-pointing process, namenode should be in safe mode, that means it will place a lock on edits\_inprogress\_xxxx file in order to create a new fsimage file or namespace image. In this process(in safe mode), client cannot write the file or perform modification to the existing blocks on HDFS, …etc. But client can read the data from datanodes because metadata information is available in memory. So namenode can provide access to the client to read the data available on datanodes.

**Why can’t we perform write/modification operations on**[HDFS](http://data-flair.training/blogs/comprehensive-hdfs-guide-introduction-architecture-data-read-write-tutorial/)**when namenode is in safe mode?**

As said earlier that when client want to write a file or perform modifications to an existing blocks on HDFS, first of all namenode has to write a transaction record in edits\_inprogress\_xxxx file, but this file is already locked for check-pointing process. Then how namenode can serve the clients write/modifications requests. That’s why namenode is not responsible to perform check-pointing process. There is secondary namenode/ checkpoint node to take care of check-pointing process.

Along with issue, there is one issue, that is memory space. To perform check-pointing process, it requires more memory in order create a new fsimage.

That’s why we need a separate node for secondary namenode/checkpoint node.

From our end also, we can perform check-pointing.

**Use below command to enter into safemode.**

hdfs dfsadmin -safemode enter

hdfs dfsadmin -saveNamespace

When you fire above command, then the check-pointing process will get started. Before doing this, you need to bring the node to safemode. Once you are done with check-pointing, then node should leave the safemode.

**To leave the safemode**

hdfs dfsadmin -safemode leave

**With below command, you can also check whether node is which mode.**

**hdfs dfsadmin -safemode get**

* Q: Where does NameNode keeps meta data? does it store the same in hdfs or local fs or memory and why ?

Ans:

**In**[Hadoop](http://data-flair.training/blogs/hadoop-introduction-tutorial-quick-guide/)**, Namenode has two types of files.**

1) edit logs files  
2) FsImage files  
These files are available on namenode disk(persistent data storage).

when you are starting namenode, latest fsimage file is load into **“in-memory”**. and at the same time, edit log file is also loaded into memory if fsimage file doesn’t contain up-to date information. The information which is available in edit log(s) will be replayed to update the in-memory of fsimage data. What information is available in editlog, fsimage…etc can be shown as below.

Namenode stored metadata in “in-memory” in order to serve the multiple client request(s) as fast as possible. If we didn’t stored the metadata information in “in-memory”, then for every operation, namenode has to load the metadata information from the disk to in-memory and start performing various check’s on this metadata information. This process will consume more disk seek time for every operations(Reading from and Writing to Disk is a time consuming process. That’s why metadata information is stored in “In-Memory”. As part of in-memory, it will have both file metadata and bitmap metadata information.

**In in-memory, it contains two types/forms of metadata.**  
1) File metadata  
2) BitMap metadata

File Metadata: This contains file name, permission, owner, group, others, replication, [Block](http://data-flair.training/blogs/data-blocks-hdfs-hadoop-distributed-file-system/)ids[b1,b2,b3..etc: These block ids are unique across the cluster], file size, ..etc. File metadata contains only file to block mapping information. But block to datanode mapping is not available in File Metadata. For each file metadata, it consumes 150 bytes in “in-memory”. For each block metadata, it consumes 150 bytes is used in “in-memory”.

Block to datanode(s) mapping is available in BitMap metadata. This contains information like Block-ID(s), address of datanode(s), generated timestamp, block length,…etc. BitMap is stored only in “in-memory”. This information is not stored in disk (persistent disk). How can we build this information when we restart the namenode?. Namenode will be in safe-mode after restart was happened. At this time, namenode will wait for signal from datanodes(s). These will send the block report to namenode. With this information namenode will build the BitMap metadata.

Based on these two metadata(s), namenode can serve the client request.

Let’s say, i want to write a file”empolyee.txt(300 MB file, replication factor :3)” file on HDFS slave nodes. First off all client need to interact with namenode. Namenode checks the whether the file is already exist or not. Where the namenode check?. Namenode will check in “in-memory” file metadata information. If it is not available, then namenode maintain a record in edit log. After record has written into edit log, namenode start writing the metadata information of this file int “in-memory”. Namenode doesn’t write the file metadata information in fsimage file when the cluster is up and running. For each operation/record, an incremental counter value is placed. Inside the hadoop, writing a file is not a single operation. There involves lot of transations (like replicating the block once first block has written onto first datanode…etc)

when you convert the binary format of editlog file(s) to xml format. you can see how many transaction(s) has been performed for a single file. For each transaction, a new record will be maintain. For each record, a transaction ID will be assigned.

hdfs oev -i edit.\_xxxxxxxxx -o edit\_myformat.xml -p XML

open edit\_myformat.xml file and see the internal information of each every transaction has been done.

**what is fsimage and edit log?**

As said earlier, firstly the information is stored in edit logs, then after in-memory of fsimage file got updated(Only in-memory, not in disk). In general, fsimage file information is equal to in-memory file metadata. How fsimage(disk) contain latest information?. This could be achieved by using checkpoint node/secondary node.

I want to write/store empdept.txt on [HDFS](http://data-flair.training/blogs/comprehensive-hdfs-guide-introduction-architecture-data-read-write-tutorial/).

Here we want to store file on HDFS. To indicate that whether it is a file or a directory. This information is consider as a metadata. To differentiate whether it is a file or directory, it uses FILE as value for file, DIRECTORY as value for directory. Look at the below open and end type tag.

<type>FILE</type>

Next the file name is consider as metadata. To indicate, empdept.txt as filename, it used open and end name tag.  
Next the replication tag which indicates that we have used one (1) for this file particular. I’m in pseudo distribute mode. So we had configured one for replication factor. This information is also a metadata of that particular file. and so on….etc

<name>empdept.txt</name>

<replication>1</replication>

<mtime>1498301552634</mtime>

<atime>1498301551522</atime>

<perferredBlockSize>134217728</perferredBlockSize>

<permission>hdadmin:supergroup:rw-r--r--</permission>

In in-memory, you can see below information also (My input file size = 51 bytes)  
id: This indicates the block id which is unique across the cluster  
numBytes: This indicates that how many bytes of that particular block has.

<blocks>

<block><id>1073741825</id>

<genstamp>1001</genstamp>

<numBytes>51</numBytes>

</block>

</blocks>

In the above, 1073741825 number should be same as part of the block name(blk\_1073741825) on data node. Look at the below(My data node location).

hdadmin@ubuntu:~/hdata/dfs/data/current/BP-2145751794-127.0.1.1-1498301240754/current/finalized/subdir0/subdir0$ l  
blk\_1073741825 blk\_1073741825\_1001.meta

In summary about file metadata (in-memory),

<INodeSection><lastInodeId>16386</lastInodeId><inode><id>16385</id>

<type>DIRECTORY</type>

<name></name>

<mtime>1498301552643</mtime>

<permission>hdadmin:supergroup:rwxr-xr-x</permission>

<nsquota>9223372036854775807</nsquota>

<dsquota>-1</dsquota>

</inode>

<inode>

<id>16386</id>

<type>FILE</type>

<name>empdept.txt</name><replication>1</replication>

<mtime>1498301552634</mtime><atime>1498301551522</atime>

<perferredBlockSize>134217728</perferredBlockSize>

<permission>hdadmin:supergroup:rw-r--r--</permission>

<blocks>

<block><id>1073741825</id>

<genstamp>1001</genstamp>

<numBytes>51</numBytes>

</block>

</blocks>

</inode>

</INodeSection>

For each file,

**type tag:**To indicate whether it is a file or a directory.  
**name tag:** file name  
**replication tag:**replication factor for that particular file  
**perferredBlockSize tag:** default block size  
**permission:**permission for that client  
**blocks:**we may have one or more no of blocks for a particular file. All block(s) will be part of blocks tag.  
**block:** It contains ID (id tag), generated timestamp(genstamp tag), number of bytes(numBytes tag).

All the above information from type tag to blocks tag will be the part of inode tag.

So in in-memory, it contains only file metadata information. This file metadata information is persistently stored on disk by using check point node or secondary node. Block to datanode mapping is also available in “in-memory”. This metadata is not stored persistently. As i already stated how this could be built. I will talk about secondary node, checkpoint node,

* + Name node stores metadata of the distributed file system such as file to [Block](http://data-flair.training/blogs/data-blocks-hdfs-hadoop-distributed-file-system/) mapping, location of blocks on data node, active data nodes, file permissions, owner of files etc.

This is most critical piece of software in entire [HDFS](http://data-flair.training/blogs/comprehensive-hdfs-guide-introduction-architecture-data-read-write-tutorial/) file system, Name node is the first point of contact for any client to perform read and write operations to get metadata and then it can perform file I/O directly with actual data nodes.

[Read-write operation](http://data-flair.training/blogs/hadoop-hdfs-data-read-and-write-operations/) will be effective and optimized only when we get to access the metadata quickly, to achieve speed and simplicity NN stores all information in main memory

Name node also stores a snapshot of entire metadata states into the local disk as fsimage file, whenever we start NN this copy will be brought into main memory, this helps in overcoming metadata lost problem.

Q: Corresponding to a file of 1GB, how much Metadata Name Node will store. Block-size, Replication factor will be default.?

Ans:

When a file is put into [HDFS](http://data-flair.training/blogs/comprehensive-hdfs-guide-introduction-architecture-data-read-write-tutorial/), it is split into blocks (of configurable size).  
Let’s say we have a file called **“file.txt”** that is 1GB (1000MB) and our block size is 128MB. We will end up with 7 128MB blocks and a 104MB block. The NameNode keeps track of the fact that “file.txt” in HDFS maps to these eight blocks and three replicas of each block. DataNodes store blocks, not files, so the mapping is important to understanding where our data is and what our data is.

Corresponding to a block 150 bytes (roughly) of metadata is created, Since there are 8 blocks with replication factor 3 i.e. 24 blocks. Hence 150×24 = 3600 bytes of metadata will be created.

On disk, the NameNode stores the metadata for the file system. This includes file and directory permissions, ownerships, and assigned blocks in the fsimage and the edit logs. In properly configured setups, it also includes a list of DataNodes that make up the HDFS (dfs.include parameter) and DataNodes that are to be removed from

that list (dfs.exclude parameter). Note that which DataNodes have which blocks is only stored in memory and not on disk.

Q: when hadoop / namenode enter in the safe mode ? what is the need of safemode ? why namenode enters into the safemode

Ans: When Hadoop NameNode enter in the Safemode ?

The Namenode enters Safemode in primarily two cases:

During the startup of namenode daemon, the namenode enters safe mode for a certain period of time.  
The administrator can also enter the Safemode manually with the below command (in the case of maintenance/up-gradation of the cluster ):  
hadoop dfsadmin -safemode enter

What is the need of Safemode ?/ why namenode enters into the Safemode?

The primary reason of namenode to enter Safemode is to ensure that the minimum replication condition is reached. The minimum replication condition is when 99.9% of the blocks in the whole HDFS meet their minimum replication factor/level (which is by default 1, and can be configured by dfs.replication.min)  
This ensures High Availability, reliability, Fault Tolerance of data before any data manipulation in file system takes place.

Q: Explain data block placement policies. On what factors blocks are distributed in hdfs?

A:   
The strategy by which [Hadoop](http://data-flair.training/blogs/hadoop-introduction-tutorial-quick-guide/) distributes [Data Blocks](http://data-flair.training/blogs/data-blocks-hdfs-hadoop-distributed-file-system/)across clusters is based on trade offs between data reliability, write bandwidth and read bandwidth.

* The basic placement policy tries to place the first block on the client (in case client is in a different cluster, random datanodes are chosen other than busy/fully loaded ones)
* The second replica is placed on a different rack than the first one (also known as off rack)
* The third replica is placed on the same rack as the second one but in a different datanode.

Further replicas are placed in random nodes avoiding same rack placement.

Apart from the basic policy, hadoop also has a [Balancer](http://data-flair.training/blogs/hadoop-hdfs-disk-balancer/) daemon which distributes blocks by moving them from over-utilized datanodes to under-utilized datanodes keeping the basic placement policy in mind.

Q: The default data block size of HDFS/Hadoop is 64MB. The block size in disk is generally 4KB. Why block size is large in HDFS/Hadoop?

A: Block is the smallest unit of data that the file system stores. HDFS stores each file as blocks. In Hadoop 1.x default Block size was 64 MB which can be configured as per the requirement. The files were splitted into 64MB blocks and then stored into the Hadoop filesystem. The hadoop applications were responsible for distributing the data block across the multiple nodes.  
All the blocks of the file are of the same size except the last block, which can be either the same or smaller.

Why blocks size 64 MB?  
HDFS have huge data sets, i.e. terabytes and petabytes of data. So like Linux file system which have 4 KB block size if we had block size 4KB for HDFS, then we would be having too many blocks and therefore too much of metadata. Managing this huge number of blocks and metadata will create huge overhead. This is something which we don’t want.

Hadoop 2.x has default block size 128MB.

Reasons for the increase in Block size from 64MB to 128MB are as follows:

To improve the NameNode performance.  
To improve the performance of MapReduce job since the number of the mapper is directly dependent on Block size.  
If we are managing a cluster of 1 petabytes and block size is 64 MB, then 15+million blocks will create which is difficult for NameNode to manage. So block size is increased from 64MB to 128MB.  
Follow the link to learn more about Data Block in Hadoop

Q: What is meant by fault tolerance in HDFS?  
How Fault Tolerance is achieved in Hadoop?

A; In **Hadoop** Failure of one node doesn’t affect accessing ([read-write operation](http://data-flair.training/blogs/hadoop-hdfs-data-read-and-write-operations/)) of data in datanode. Multiple copies of same [Block](http://data-flair.training/blogs/data-blocks-hdfs-hadoop-distributed-file-system/) will be available in other datanode, So failure of one node will not impact our work and we can make use of block from other datanode when one of the datanode(slaves) fails.

Using **Replication Factor** we can achieve to make multiple block into datanode. By default the replication factor is 3 in [HDFS](http://data-flair.training/blogs/comprehensive-hdfs-guide-introduction-architecture-data-read-write-tutorial/). But you can increase the replication as per your requirement.We can change the replication factor in hdfs-site.xml which is located under “<HADOOP-HOME>/etc/hadoop” directory.

Also we can change replication factor in command line using hdfs command “hdfs dfs -setrep -R -w 3 hadoop-env.sh”

hadoop-env.sh ->Located in “<HADOOP-HOME>/etc/hadoop”

# q: what happens if the block on HDFS is corrupted?

A: [HDFS](http://data-flair.training/blogs/comprehensive-hdfs-guide-introduction-architecture-data-read-write-tutorial/) is a very robust, therefore in order to recover corrupted [Data Block](http://data-flair.training/blogs/data-blocks-hdfs-hadoop-distributed-file-system/) it has provided the user with below:-

1. HDFS fsck (For Data Node)  
2. Namenode -recover

**DataNode:-**

Fsck is an offline process which examines on-disk structures and usually offers to fix them if they are damaged.

HDFS has its own fsck command, which you can access by running “hdfs fsck.” HDFS fsck determines which files contain corrupt blocks, and gives you options about how to fix them.

HDFS fsck command operates only data, not on metadata.This difference is irrelevant on the local filesystem because data and metadata are stored in the same place. However, for HDFS, metadata is stored on the NameNode, whereas data is stored on the DataNodes.

**NameNode :-**

When properly configured, HDFS is robust, because it stores multiple copies of everything. The administrator has the capability to recover a partial or corrupted edit log. This new functionality is called manual NameNode recovery.

Similar to fsck, NameNode recovery is an offline process. An administrator can run NameNode recovery to recover a corrupted edit log.

Sart the NameNode with the -recover flag to activate recovery mode, like:

./bin/hadoop namenode -recover

Though Manual Recovery is the Best Choice. If there is another valid copy of the edit log somewhere else, it is preferable to use that copy rather than trying to recover the corrupted copy. This is a case where [High availability](http://data-flair.training/blogs/hadoop-hdfs-namenode-high-availability/) can help a lot. If there is a standby NameNode ready to take over, there should be no need to recover the edit log on the primary.When there is no other copy of the edit log available, Manual recovery is a good choice.

**Conclusion**

The best recovery process is the one that you never need to do. High availability, combined with edit log failover, should mean that manual recovery is almost never necessary. However, it’s good to know that HDFS has tools to deal with whatever comes up.

Q: What should be the block size in hadoop cluster?

A: [Data Block](http://data-flair.training/blogs/data-blocks-hdfs-hadoop-distributed-file-system/) in [HDFS](http://data-flair.training/blogs/comprehensive-hdfs-guide-introduction-architecture-data-read-write-tutorial/) is a continuous location on the hard drive where data is stored, in general, FileSystem store data as a collection of blocks. In a similar way, HDFS stores each file as blocks, and distributes it across the [Hadoop cluster](http://data-flair.training/blogs/installation-hadoop-3-x-ubuntu-pseudo-distributed-mode/).  
**Block Size**  
There is no as such rule set by [Hadoop](http://data-flair.training/blogs/hadoop-introduction-tutorial-quick-guide/) to the bound user with certain block size. Usually, it depends on the input data. If you want to maximize throughput for a very large input file, using very large blocks (may be 128MB or even 256MB) is best. But on the other hand for smaller files, using a smaller block size is better.

So, here we are dealing with larger file large block & smaller file small blocks. In Industry we can get files of different sizes & we can have files with different block sizes on the same file system. This situation is overcome by using ”dfs.block.size” parameter when the file is written. It will help you in overriding default block size written in hdfs-site.xml

Q: What is HDFS Federation?  
How it helps namenode in scaling ?

A: Before starting HDFS Federation, let us first discuss  
**Scalability**  
The primary benefit of Hadoop is its Scalability.One can easily scale the cluster by adding more nodes.  
There are two types of Scalability in Hadoop: Vertical and Horizontal  
**Vertical scalability**  
It is also referred as “scale up”. In vertical scaling, you can increase the hardware capacity of the individual machine. In other words, you can add more RAM or CPU to your existing system to make it more robust and powerful.

**Horizontal scalability**  
It is also referred as “scale out” is basically the addition of more machines or setting up the cluster. In horizontal scaling instead of increasing hardware capacity of individual machine you add more nodes to existing cluster and most importantly, you can add more machines without stopping the system. Therefore we don’t have any downtime or green zone, nothing of such sort while scaling out. So at last to meet your requirements you will have more machines working in parallel.

**To learn more about the Scalabilty follow:** [HDFS Scalability](http://data-flair.training/blogs/features-hadoop-hdfs-overview-beginners/)

HDFS has two main layers:-

1.**Namespace** – manages directories, files and blocks. It supports file system operations such as creation, modification, deletion and listing of files and directories.

2. **Block Storage**– Block storage provides operations like creation, deletion, modification and getting the location of the blocks. It also takes care of replica placement and replication.

**Architecture without HDFS Federation**

Datanode can be scaled both vertically & horizontally. But namenode was scaled only vertically not horizontally. Architecture without HDFS Federation has multiple datanodes, but it has only one NameNodefor (one namespace) for all datanodes .This limits the number of blocks, files, and directories supported on the file system.

To overcome this limitation HDFS Federation is introduced.

**Architecture with HDFS Federation**

In order to scale the namenode horizontally, federation uses multiple independent Namenodes/namespaces. In HDFS Federation, Namenodes does not require coordination with each other as the namenode is independent. And in HDFS federation, all the datanodes are used as common storage for blocks by all the Namenodes. In HDFS Federation, each datanode in registers with all the Namenodes in the cluster. Blocks that belong to a single namespace are called Block pool. Datanodes store blocks for all the block pools in the cluster.  
Therefore, with HDFS Federation previous limitation has been overcome where we were not able to scale namenode horizontally. This also provides the scope for absolute isolation.

Q: We can set replication factor property globally in hdfs-site.xml, but for some specific files how to specify different replication factor (which is already stored in HDFS)?

A: The replication factor is a property that can be set in the HDFS configuration file( hdfs-site.xml).This will be to set global replication factor for the entire cluster.This will only work on the newly created files but not on the existing files.

Change the replication factor on a per-file basis :  
hadoop fs –setrep –w 3 /file/filename.xml

-setrep commnad to change the replication factor for files that already exist in HDFS.-R flag would recursively change the replication factor on all the files

eg:  
hadoop fs –setrep –w 3 -R /directory/dir.xml

Q: What is different between the split and block in Hadoop?  
Comparison between MapReduce InputSplit vs HDFS Block?  
Comparison between Split Size vs Block Size in Hadoop?

A: [InputSplit](http://data-flair.training/blogs/inputsplit-in-hadoop-mapreduce/) is the logical division of data.  
[Data Block](http://data-flair.training/blogs/data-blocks-hdfs-hadoop-distributed-file-system/) is the physical division of data.

In [HDFS architecture](http://data-flair.training/blogs/hadoop-hdfs-architecture/) there is concept of blocks. A typical size of HDFS block is 128MB. A large file in HDFS is broken down into chunks.  
Suppose we have 1GB of the file, and we want to place this file in HDFS then there will be 1GB/128MB= 8 Blocks and here blocks are distributed across different datanodes based on configuration.

Inputsplit is basically used during data processing in [MapReduce Program](http://data-flair.training/blogs/hadoop-mapreduce-introduction-tutorial-comprehensive-guide/). It is user defined value and can choose the size based on the size of data and how you are processing. If the user does not define the inputsplit then based on the number of blocks, then based on no of blocks input split size is considered.  
No of input splits are equal to the number of [Mappers](http://data-flair.training/blogs/mapper-in-hadoop-mapreduce/) in the program to process the data.

**For example,**  
If you have 200MB file and HDFS default block size is 128MB. Then it is chopped into 2 blocks(128MB,72MB). If you have not defined Inputsplit size then by it takes size as 2 (as there are 2blocks) and assigns 2 mappers. But if you have specified the split size as 200MB then both blocks will be considered as the single split for map reduce program and assigns one mapper.

Q: Can multiple clients write into an HDFS file concurrently?

A: No, multiple clients cannot write into an [HDFS](http://data-flair.training/blogs/hadoop-hdfs-introduction-architecture-features-operations-tutorial/) file at same time.  
When one client is given permission by Name node to write data on data node block, the block gets locked till the write operations is completed. If some other client requests to write on the same block of a particular file in data node, it is not permitted to do so. It has to wait till the write lock is revoked on a particular data node. All the requests are in the queue and only one client is allowed to write at a time.

Q: In HDFS,can we read from the file which is already open for writing?  
What happens, when client try to read from file already opened for writing in HDFS?

A: Yes, the client can read the file which is already opened for writing.  
But, the problem in reading a file which is currently being written, lies in the consistency of data i.e. [Hadoop HDFS](http://data-flair.training/blogs/comprehensive-hdfs-guide-introduction-architecture-data-read-write-tutorial/) does not provide the surety that the data which has been written into the file will be visible to a new reader before the file has been closed.

For this, one can call the **hflush operation** explicitly which will push all the data in the buffer into write pipeline and then the hflush operation will wait for acknowledgments from the datanodes. Hence, by doing this, the data that has been written into the file before the hflush operation visible to the reader for sure.

Q: Who divide the file into block while storing inside hdfs?

A: [HDFS](http://data-flair.training/blogs/comprehensive-hdfs-guide-introduction-architecture-data-read-write-tutorial/)client will divide the file(s) into [Blocks](http://data-flair.training/blogs/data-blocks-hdfs-hadoop-distributed-file-system/). HDFS client resides on client node.

Q: How indexing is done in Hadoop?

A: In Distributed file system like HDFS, indexing is diffenent from that of local file system. Here indexing and searching of data is done using the memory of the HDFS node where data is residing.

The generated index files are stored in a folder in directory where the actual data is residing. Searching is similar to the search in local file system, but RAM directory object is used and is done on the index file residing in the Memory.

Indexing in Hadoop depends on the block size, The last part of the data stored on hdfs will contain the details about the storage of next pat of the Block.

Q: What is rack awareness in Hadoop HDFS?  
What is the need of Rack awareness?

A: **Rack**– It the collection of machines around 40-50. All these machines are connected using the same network switch and if that network goes down then all machines in that rack will be out of service. Thus we say rack is down.

[Rack Awareness](http://data-flair.training/blogs/rack-awareness-hadoop-hdfs/) was introduced by [Apache Hadoop](http://data-flair.training/blogs/hadoop-introduction-tutorial-quick-guide/) to overcome this issue. In Rack Awareness, NameNode chooses the DataNode which is closer to the same rack or nearby rack. NameNode maintains **Rack ids** of each DataNode to achieve rack information. Thus, this concept chooses Datanodes based on the rack information. NameNode in hadoop makes ensures that all the replicas should not stored on the same rack or single rack. Rack Awareness Algorithm reduces latency as well as [Fault Tolerance](http://data-flair.training/blogs/learn-hadoop-hdfs-fault-tolerance/).

Default replication factor is 3. Therefore according to Rack Awareness Algorithm:

* The first replica of the block will store on a local rack.
* The next replica will store on another datanode within the same rack.
* The third replica stored on the different rack.

**In Hadoop, we need Rack Awareness for below reason: It improves**

* Data high availability and reliability.
* The performance of the cluster.
* Network bandwidth.

Q: How to create Hadoop Archive (.HAR) ? When should we create Hadoop archive ?

A: **Hadoop archives** are created to deal with small files problem.

**Small files problem**:  
As we know that [Hadoop](http://data-flair.training/blogs/hadoop-introduction-tutorial-quick-guide/) is designed to deal with large files and the namenode keeps the metadata of the files in its memory.  
But what if we give it small files, that too in large number; After splitting the data the namenode has to keep too many records in its memory, which will make it inefficient.  
So, to tackle this problem, we create Hadoop archives. Hadoop archives files pack the [HDFS](http://data-flair.training/blogs/comprehensive-hdfs-guide-introduction-architecture-data-read-write-tutorial/) files into archives and we can directly use these files an as input to the [MapReduce](http://data-flair.training/blogs/hadoop-mapreduce-introduction-tutorial-comprehensive-guide/) jobs.

The command for it is:

$ hadoop archive -archiveName myArch.har /data

If we list the archive file:

$hadoop fs -ls /data/myArch.har

/data/myArch..har/\_index

/data/myArch..har/\_masterindex

/data/myArch..har/part-0

where part files are the original files concatenated together with big files and index files are to look up for the small files in the big part file.

**But even Hadoop archive files have few limitations. They are:**

* These files are unmodifiable, i.e., once you archive them, they cannot be modified.
* They take space as much as the original files
* When they are given as input to the MapReduce jobs, the small files are processed individually by a mapper which is inefficient.

Q: How is Data Integrity achieved in HDFS?  
Is Data Integrated in HDFS?

A:   
1) Data Integrity means to make sure that no data is lost or corrupted during storage or processing of the Data.

2) Since in Hadoop, amount of data being written or read is large in Volume, chances of data corruption is more.

3) So in Hadoop checksum is computed when data written to the disk for the first time and again checked while reading data from the disk. If checksum matches the original checksum then it is said that data is not corrupted otherwise it is said to be corrupted.

4) Its just data detection error.

5) It is possible that it’s the checksum that is corrupt, not the data, but this is very unlikely, because the checksum is much smaller than the data

6) HDFS uses a more efficient variant called CRC-32C to calculate checksum.

7) DataNodes are responsible for verifying the data they receive before storing the data and its checksum. Checksum is computed for the data that they receive from clients and from other DataNodes during replication

8) Hadoop can heal the corrupted data by copying one of the good replica to produce the new replica which is uncorrupt replica.

9) if a client detects an error when reading a block, it reports the bad block and the DataNodes it was trying to read from to the NameNode before throwing a Checksum Exception.

10) The NameNode marks the block replica as corrupt so it doesn’t direct any more clients to it or try to copy this replica to another DataNodes.

11) It provides copy of the block another DataNodes which is to be replicated, so its replication factor is back at the expected level.

12) Once this has happened, the corrupt replica is deleted

Tls :

to setup tls/ssl we require keys to present into each format jks and pem.we use a combination of java key tool and openssl for creating rootCA and internal certificates.

once certificates are created then we have converted them into pem format using pkcs12 format using openssl.While TlS/SSL connection negotiation client talk to truststore,that is why

save rootca.pem into jsscerts so that clients can verify the authenticity of the server certificate while connecting to the server.

Tls levels:

* Level 1 (Good) - This level only configures encrypted communication between the browser and Cloudera Manager, and between Agents and the Cloudera Manager Server. See [Configuring TLS Encryption Only for Cloudera Manager](https://docs.cloudera.com/documentation/enterprise/5-2-x/topics/cm_sg_tls_browser.html#xd_583c10bfdbd326ba-7dae4aa6-147c30d0933--7a61) followed by [Level 1: Configuring TLS Encryption for Cloudera Manager Agents](https://docs.cloudera.com/documentation/enterprise/5-2-x/topics/cm_sg_config_tls_encr.html#topic_2) for instructions. Level 1 encryption prevents snooping of commands and controls ongoing communication between the Agents and Cloudera Manager.
* Level 2 (Better) - This level includes encrypted communication between the Agents and the Server, as well as strong verification of the Cloudera Manager Server certificate by the Agents. See [Level 2: Configuring TLS Verification of Cloudera Manager Server by the Agents](https://docs.cloudera.com/documentation/enterprise/5-2-x/topics/cm_sg_config_tls_auth.html#topic_3). Level 2 provides Agents with an additional level of security by verifying trust for the certificate presented by the Cloudera Manager Server.

Level 3 (Best) - Encrypted communication between the Agents and the Server. Level 3 TLS includes encrypted communication between the Agents and the Server, strong verification of the Cloudera Manager Server certificate by the Agents and authentication of Agents to the Cloudera Manager Server using self-signed or CA-signed certs. See [Level 3: Configuring TLS Authentication of Agents to the Cloudera Manager Server](https://docs.cloudera.com/documentation/enterprise/5-2-x/topics/cm_sg_config_tls_agent_auth.html#topic_4). Level 3 addresses the untrusted network scenario where you need to prevent cluster Servers being spoofed by untrusted Agents running on a host. Cloudera recommends you configure Level 3 TLS encryption for untrusted network environments before enabling Kerberos authentication. This provides secure communication of keytabs between the Cloudera Manager Server and verified Agents across the cluster

Leve1: use \_tls =1

Level2:verify\_cert\_file=/opt/hadoop/security/ca-certs/rootCA.pem

Level3: The Cloudera Manager agent obtains the password from a text file, not from a command line parameter or environment variable. The password file allows you to use file permissions to protect the password.

1. Use a text editor to create a file called /etc/cloudera-scm-agent/agentkey.pw that contains the password. Here our password is “password”.
2. Change ownership of the file to root:

sudo chown root:root /etc/cloudera-scm-agent/agentkey.pw

1. Change the permissions of the file:

sudo chmod 440 /etc/cloudera-scm-agent/agentkey.pw

|  |  |  |
| --- | --- | --- |
| client\_key\_file | /opt/hadoop/security/x509/key.pem | Path to the private key file. |
| client\_keypw\_file | /etc/cloudera-scm-agent/agentkey.pw | Path to the private key password file. |
| client\_cert\_file | /opt/hadoop/security/x509/cert.pem | Path to the client certificate file. |

Kerberos:

krb5-sever workstation

/etc/krb5.conf ----> libdeafult ---realm name ticket lifetime: renew life time,encryption type

/var/kerberos/krbkdc ---> kadm5.acl ---we will add privileges who can access kdc server.

kdc.conf ---kdc port number :88 ,supported encp type : max renevable life

sudo krb5\_util create ---kdc database which will hold principal

sudo service krb5kdc start

sudo service kadmin start

install krb-5 workstation and distribute krb5.conf file to all hosts.

create cm/admin prinicipal for cloudera.

teset kerberos on any machine via --- kinit cm/admin

KMS:

navigator key trusty server or file based password protected java keystore --->stores keys ---KMS ----HDFS CLIENTS

/

ZONE ---> EZ ---->EDEK ---NAMENODE

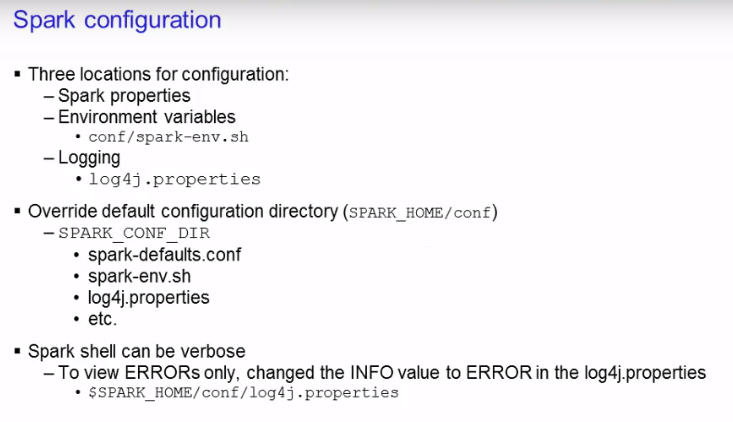
FILE ---> DEK

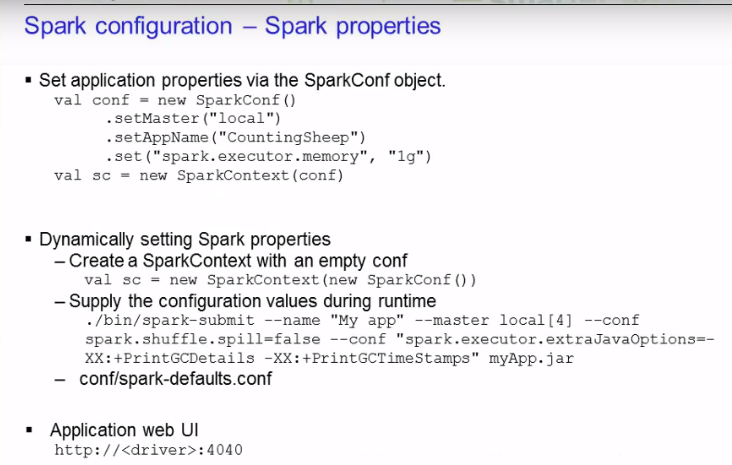
• hadoop key create mykey1

• hadoop fs -mkdir /tmp/zone1

• hdfs crypto -createZone -keyName mykey1 -path /tmp/zone1

Spark :





1- **The first method is by passing application properties via the SparkConf object**

As you know, the SparkConf variable is used to create the SparkContext object. In the example shown on this slide, you set the master node as local, the appName as **"CountingSheep",** and you allow **1GB** for **each** of the executor processes.

2- **The second method is to dynamically set the Spark properties**

Spark allows you to pass in an **empty SparkConf** when creating the SparkContext as shown on the slide.

You can then either supply the values during **runtime** by using the command line options **--master** or the **--conf.** You can see the list of options using the **--help** when executing the spark-submit script.

On the slide here, you give the app name of **My App** and telling it to run on the **local system** with **four cores.** You set the **spark.shuffle.spill to false** and the various java options at the end. Finally you supply the application JAR file after all the properties have been specified.

Another way to set Spark properties is to provide your settings inside the **spark-defaults.conf**

file. The spark-submit script will read in the configurations from this file. You can view the Spark properties on the application web UI at the port **4040** by default.

You can find a list of all the properties on the spark.apache.org website.

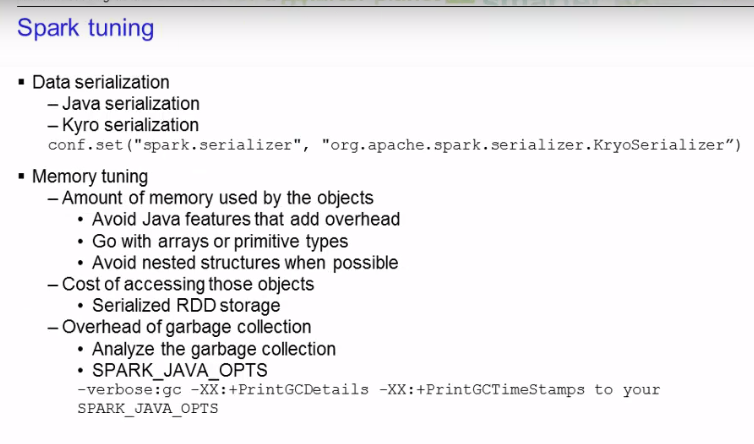
properties set directly on the SparkConf take highest precedence, then flags passed to spark-submit or spark-shell is second and finally options in the spark-defaults.conf file is the lowest priority.

**Spark tuning:**

Spark programs can be bottlenecked by any resource in the cluster. Due to Spark's nature

of the in-memory computations, **data serialization** and **memory** tuning are two areas that will

improve performance.



**Data serialization:**

Data serialization is crucial for network performance and to reduce

memory use. It is often the first thing you should look at when tuning Spark applications.

Spark provides two serialization libraries.

1- **Java serialization**

Java serialization provides a lot more flexibility, but it is quiet **slow** and leads to large serialized objects. This is the **default** library that Spark uses to serialize objects.

2- **Kyro serialization**

Kyro serialization is much quicker than Java, but does not support all Serializable types. It would require you to register these types in advance for best performance.

To use Kyro serialization, you can set it using the **SparkConf object**.

**Memory Tuning:**

With memory tuning, you have to consider **three** things. The amount of memory used by the objects

(whether or not you want the entire object to fit in memory). The cost of accessing those objects and the overhead garbage collection.

You can determine how much memory your dataset requires by creating a RDD, put it into cache, and look at the SparkContext log on your driver program. Examining that log will show you how much memory your dataset uses.

**Few tips to reduce the amount of memory used by each object.**

**1 - Try to avoid Java features that add overhead such as pointer based data structures and wrapper objects. If possible go with arrays or primitive types and try to avoid nested structures.**

**2- Serialized storage can also help to reduce memory usage. The downside would be that it**

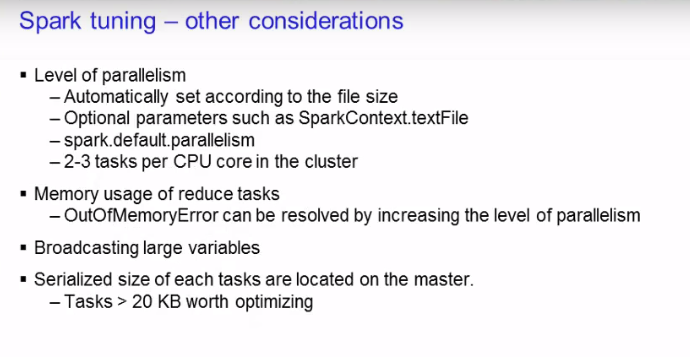
**will take longer to access the object because you have to deserialized it before you can**

**use it.**

**3-** **You can collect statistics on the garbage collection to see how frequently it occurs**

**and the amount of time spent on it. To do so, add the line to the SPARK\_JAVA\_OPTS environment**

**variable.**



**Level of paralleslism:**

The level of parallelism should be considered in order to fully utilize your cluster. It is automatically set to the **file size of the task**, but you can configure this through optional parameters such as in the **SparkContext.textFile**.

You can also set the default level in the **spark.default.parallelism** config property. Generally, it is recommended to set **2-3 tasks per CPU core in the cluster**.

Sometimes when your RDD does not fit in memory, you will get an **OutOfMemoryError**. In cases

like this, often by increasing the level of parallelism will resolve this issue. By increasing

the level, each set of task input will be smaller, so it can fit in memory.

Using Spark's capability to **broadcast large variables** greatly reduces the size of the

serialized object. A good example would be if you have some type of static lookup table.

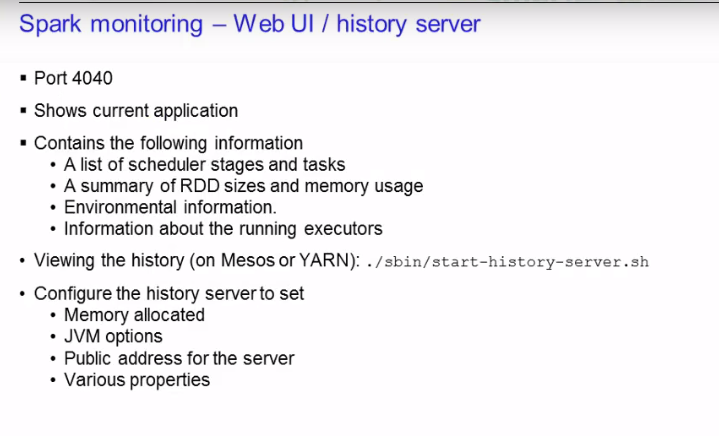
Consider turning that into a broadcast variable so it does not need to be passed on to each

of the worker nodes.

Spark prints the serialized size of each tasks in the master. Check that out to examine if

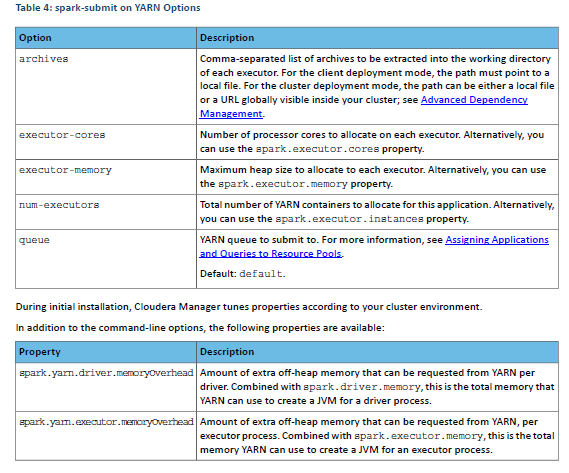
any tasks are too large. If you see some tasks larger than 20KB, it's worth taking a look

to see if you can optimize it further, such as creating **broadcast variables**.



all application history data resides under hdfs in below dir:

[hdfs@hadoop-slave-1 ~]$ hdfs dfs -ls /user/spark/applicationHistory



Dynamic Allocation

Dynamic allocation allows Spark to dynamically scale the cluster resources allocated to your application based on the

workload. When dynamic allocation is enabled and a Spark application has a backlog of pending tasks, it can request

executors. When the application becomes idle, its executors are released and can be acquired by other applications.

Starting with CDH 5.5, dynamic allocation is enabled by default. Table 5: Dynamic Allocation Properties on page 44

describes properties to control dynamic allocation.

If you set spark.dynamicAllocation.enabled to false or use the --num-executors command-line argument

or set the spark.executor.instances property when running a Spark application, dynamic allocation is disabled.

Using Spark Streaming

Spark Streaming is an extension of core Spark that enables scalable, high-throughput, fault-tolerant processing of data

streams. Spark Streaming receives input data streams and divides the data into batches called DStreams. DStreams

can be created either from sources such as Kafka, Flume, and Kinesis, or by applying operations on other DStreams.

Every input DStream is associated with a Receiver, which receives the data from a source and stores it in executor

memory.

For detailed information on Spark Streaming, see Spark Streaming Programming Guide.

Spark Streaming and Dynamic Allocation

Starting with CDH 5.5, dynamic allocation is enabled by default, which means that executors are removed when idle.

However, dynamic allocation is not effective in Spark Streaming. In Spark Streaming, data comes in every batch, and

executors run whenever data is available. If the executor idle timeout is less than the batch duration, executors are

constantly being added and removed. However, if the executor idle timeout is greater than the batch duration, executors

are never removed. Therefore, Cloudera recommends that you disable dynamic allocation by setting

spark.dynamicAllocation.enabled to false when running streaming applications.

# Running Spark Streaming Jobs on a Kerberos-Enabled Cluster

Use the following steps to run a Spark Streaming job on a Kerberos-enabled cluster.

1. Select or create a user account to be used as principal.

This should not be the kafka or spark service account.

1. Generate a keytab for the user.
2. Create a Java Authentication and Authorization Service (JAAS) login configuration file: for example, key.conf.
3. Add configuration settings that specify the user keytab.

The keytab and configuration files are distributed using YARN local resources. Because they reside in the current directory of the Spark YARN container, you should specify the location as ./v.keytab.

The following example specifies keytab location ./v.keytab for principal vagrant@example.com:

KafkaClient {

com.sun.security.auth.module.Krb5LoginModule required

useKeyTab=true

keyTab="./v.keytab"

storeKey=true

useTicketCache=false

serviceName="kafka"

principal="vagrant@EXAMPLE.COM";

};

1. In your spark-submit command, pass the JAAS configuration file and keytab as local resource files, using the --files option, and specify the JAAS configuration file options to the JVM options specified for the driver and executor:
2. spark-submit \
3. --files key.conf#key.conf,v.keytab#v.keytab \
4. --driver-java-options "-Djava.security.auth.login.config=./key.conf" \
5. --conf "spark.executor.extraJavaOptions=-Djava.security.auth.login.config=./key.conf" \

...

1. Pass any relevant Kafka security options to your streaming application.

For example, the KafkaWordCount example accepts PLAINTEXTSASL as the last option in the command line:

KafkaWordCount /vagrant/spark-examples.jar c6402:2181 abc ts 1 PLAINTEXTSASL

Kafka:

Kafka’s log directory(/var/local/kafka/data),

root@hadoop-slave-1 358-kafka-KAFKA\_BROKER]# head -n 20 kafka.properties

auto.create.topics.enable=true

auto.leader.rebalance.enable=true

**broker.id=72**

controlled.shutdown.enable=true

controlled.shutdown.max.retries=3

**default.replication.factor=1**

**delete.topic.enable=true**

kafka.http.metrics.host=0.0.0.0

kafka.http.metrics.port=24042

kafka.log4j.dir=/var/log/kafka

kerberos.auth.enable=false

leader.imbalance.check.interval.seconds=300

leader.imbalance.per.broker.percentage=10

log.cleaner.dedupe.buffer.size=134217728

log.cleaner.delete.retention.ms=604800000

log.cleaner.enable=true

log.cleaner.min.cleanable.ratio=0.5

log.cleaner.threads=1

**log.dirs=/var/local/kafka/data**

log.retention.bytes=-1

**log.retention.check.interval.ms=300000 🡪 every 5 minutes it will check log segment for deletion according to the retention policy**

**log.retention.hours=168 -🡪 7 days**

log.roll.hours=168

**log.segment.bytes=1073741824 -🡪 log segement length eqvals to 1Gb**

message.max.bytes=1000000

**min.insync.replicas=1 --🡪put 2 for safe in production**

num.io.threads=8

**num.partitions=1**

num.replica.fetchers=1

offsets.topic.num.partitions=50

offsets.topic.replication.factor=3

port=9092

replica.fetch.max.bytes=1048576

replica.lag.max.messages=4000

replica.lag.time.max.ms=10000

sentry.kafka.caching.enable=true

sentry.kafka.caching.ttl.ms=30000

sentry.kafka.caching.update.failures.count=3

unclean.leader.election.enable=false

zookeeper.session.timeout.ms=6000

zookeeper.connect=hadoop-slave-1.us-central1-a.c.fleet-point-292913.internal:2181,hadoop-slave-2.us-central1-a.c.fleet-point-292913.internal:2181,hadoop-slave-3.us-central1-a.c.fleet-point-292913.internal:2181

kafka.metrics.reporters=nl.techop.kafka.KafkaHttpMetricsReporter

security.inter.broker.protocol=PLAINTEXT

**listeners=PLAINTEXT://hadoop-slave-1.us-central1-a.c.fleet-point-292913.internal:9092,**

broker.id.generation.enable=false

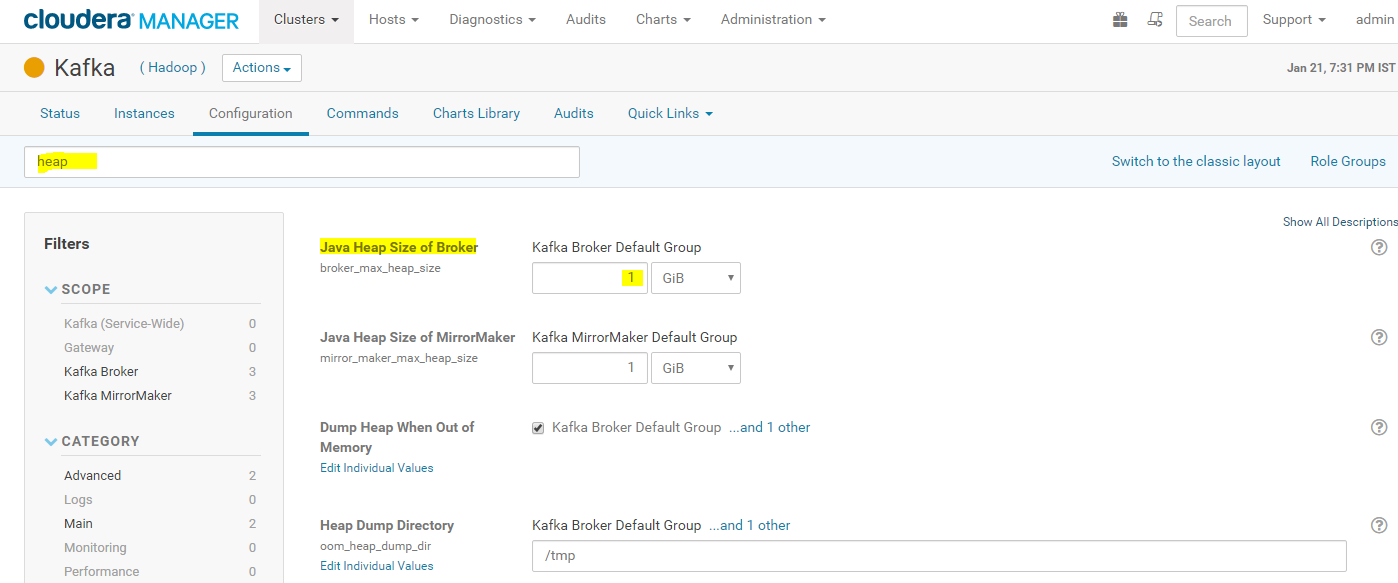
sasl.kerberos.service.name=kafka



There could be three main problems occour during kafka installation.

1- **java heap out of memory exception**

make sure that the node on which , broker instance is running should have enough memory left and accordingly assigned in “**Java Heap Size of Broker” under configuration. Otherwise java heap out of memory exception will occur .**



**2-** number two problem that can happen is if the broker was installed earlier, it used to leave the earlier data in this particular folder. that may leave inconsistency. so we can remove the earlier data and start creating the new data in kafka.

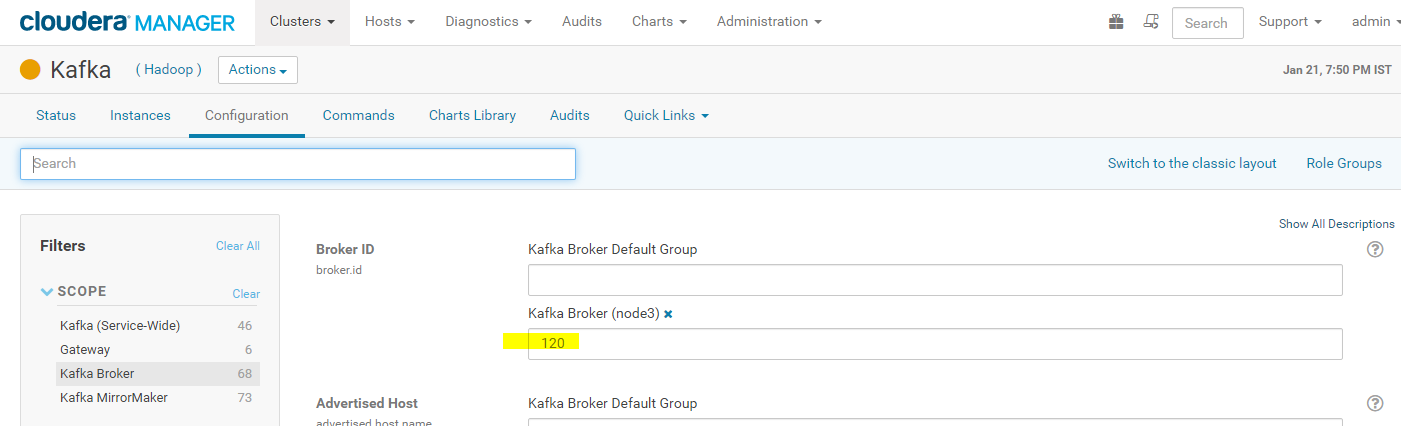
[root@node3 ~]# **cd /var/local/kafka/**

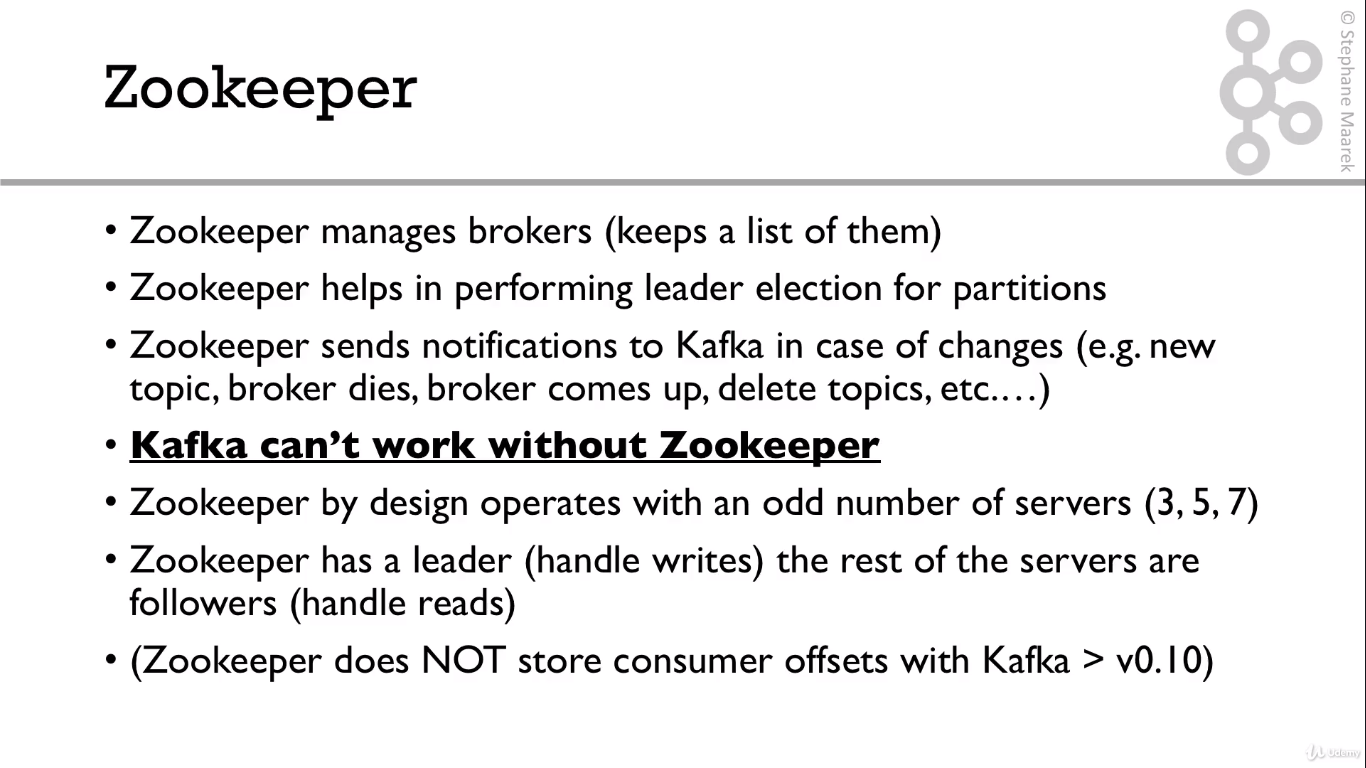
[root@node3 kafka]# ls

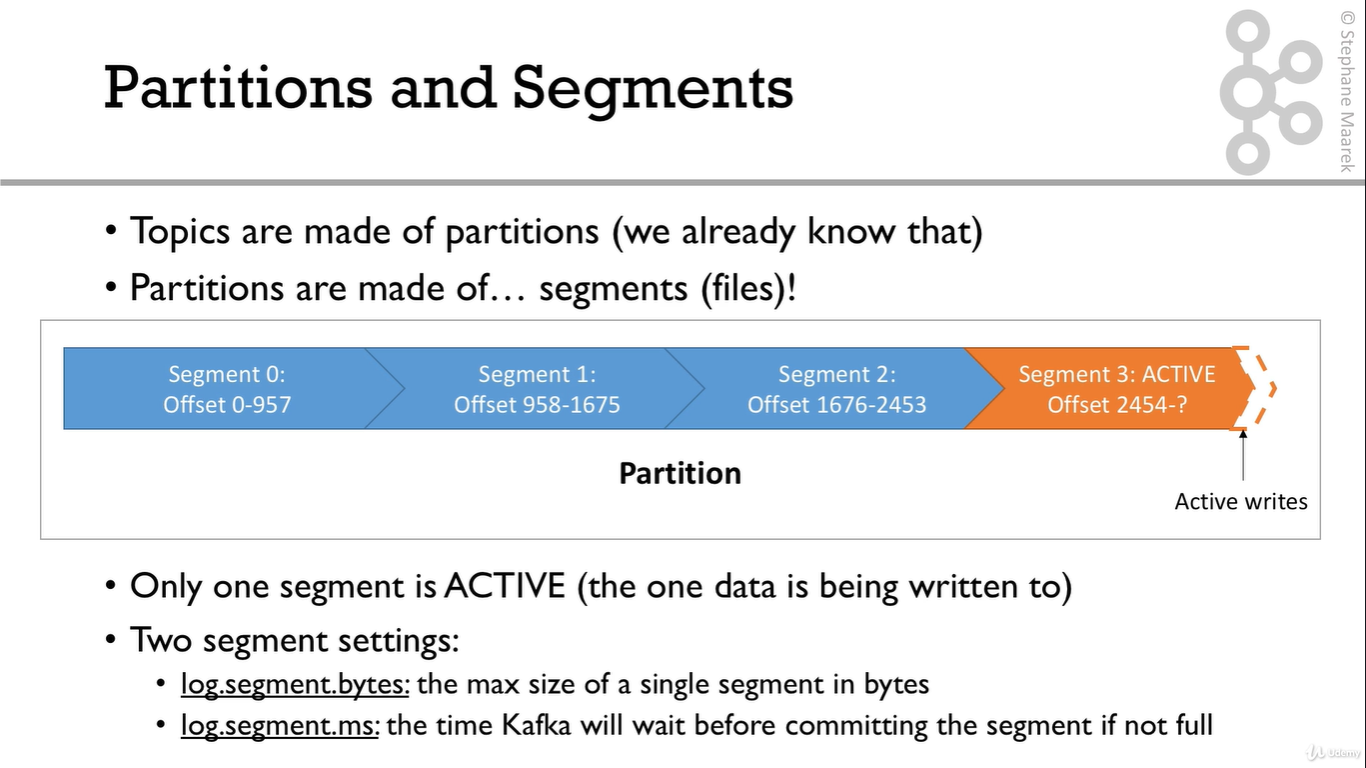
**Data**

**3-** another problem that may occur, if we leave any past configuration as it is, it would go ahead and create the broker id for that specific instance so the broker id there may be a mismatch with this particular id and the one that is configured

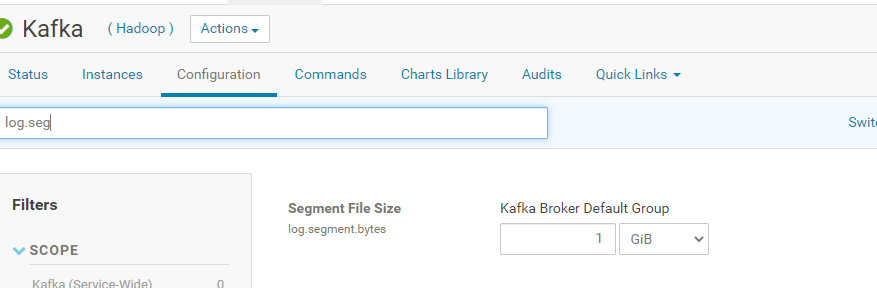
so if you change this, that should work.

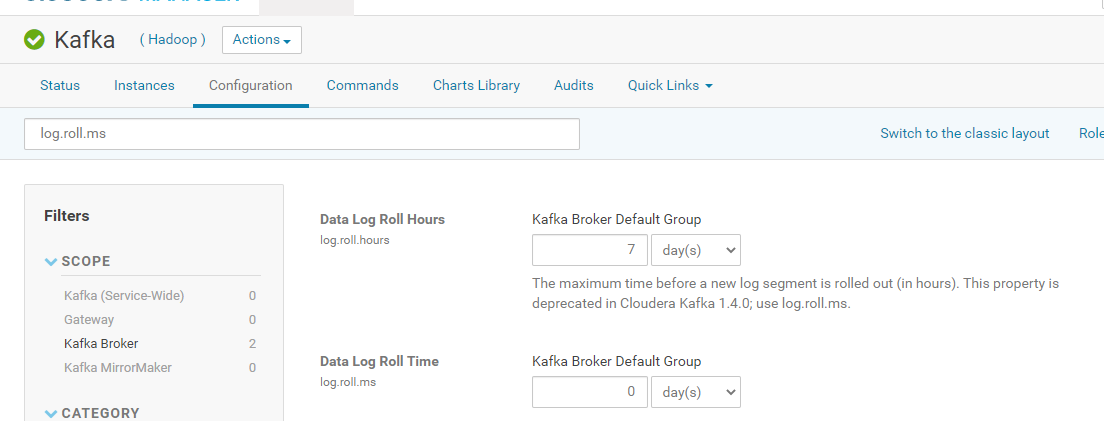


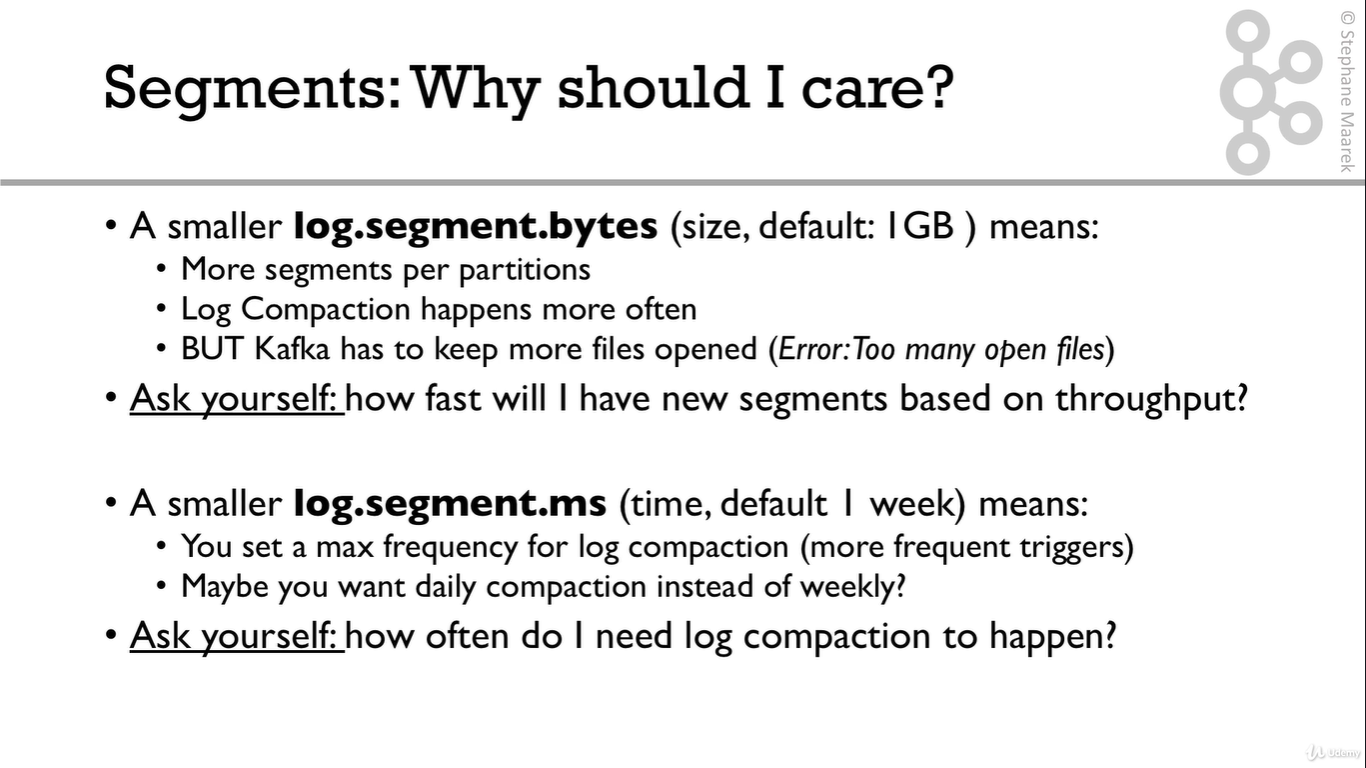


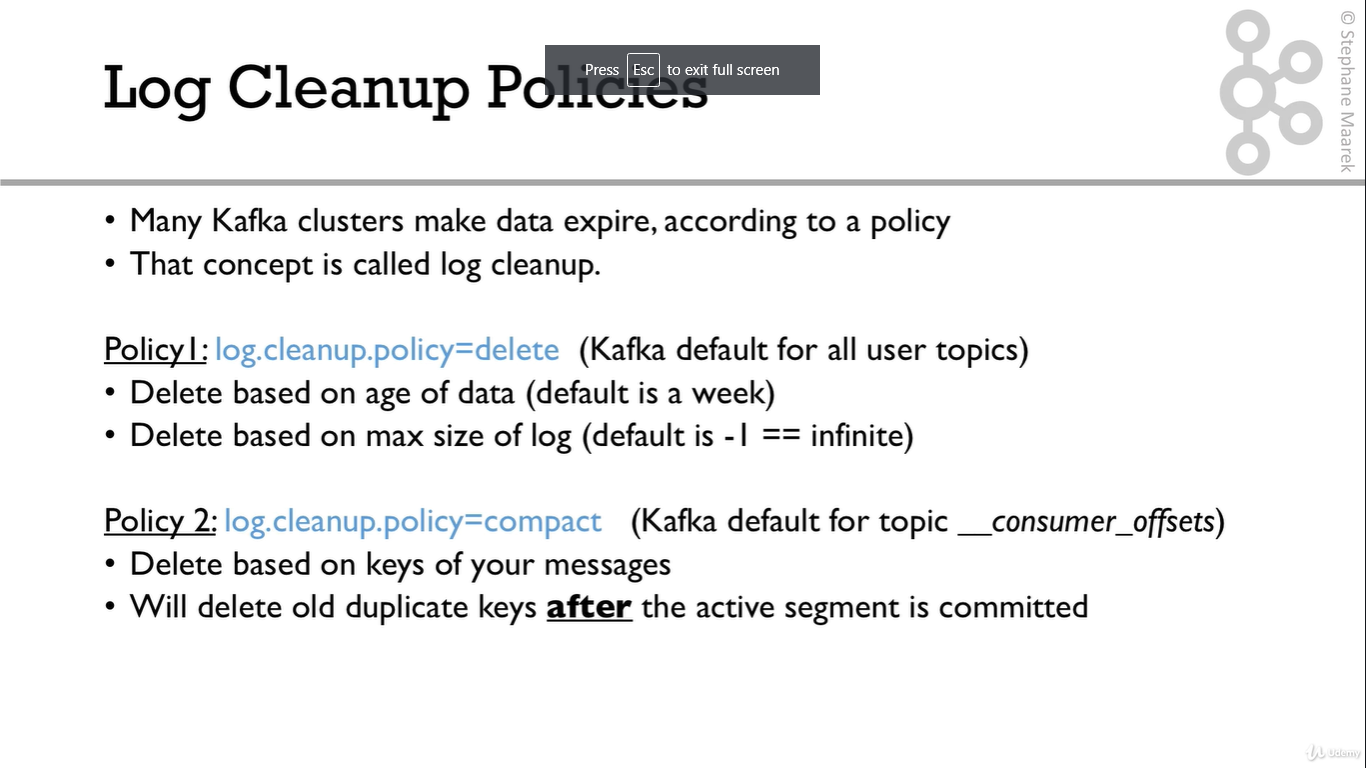


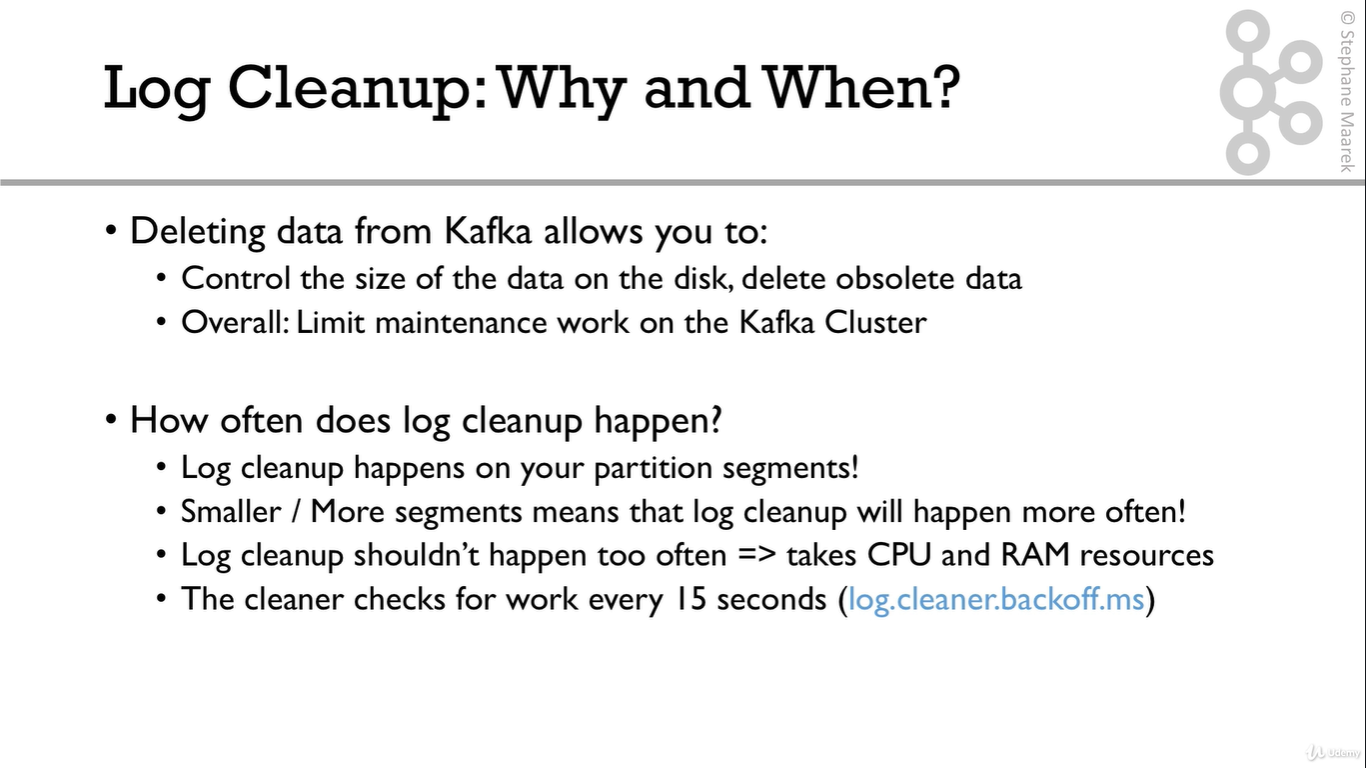
In respect of cloudera two settings are as below:

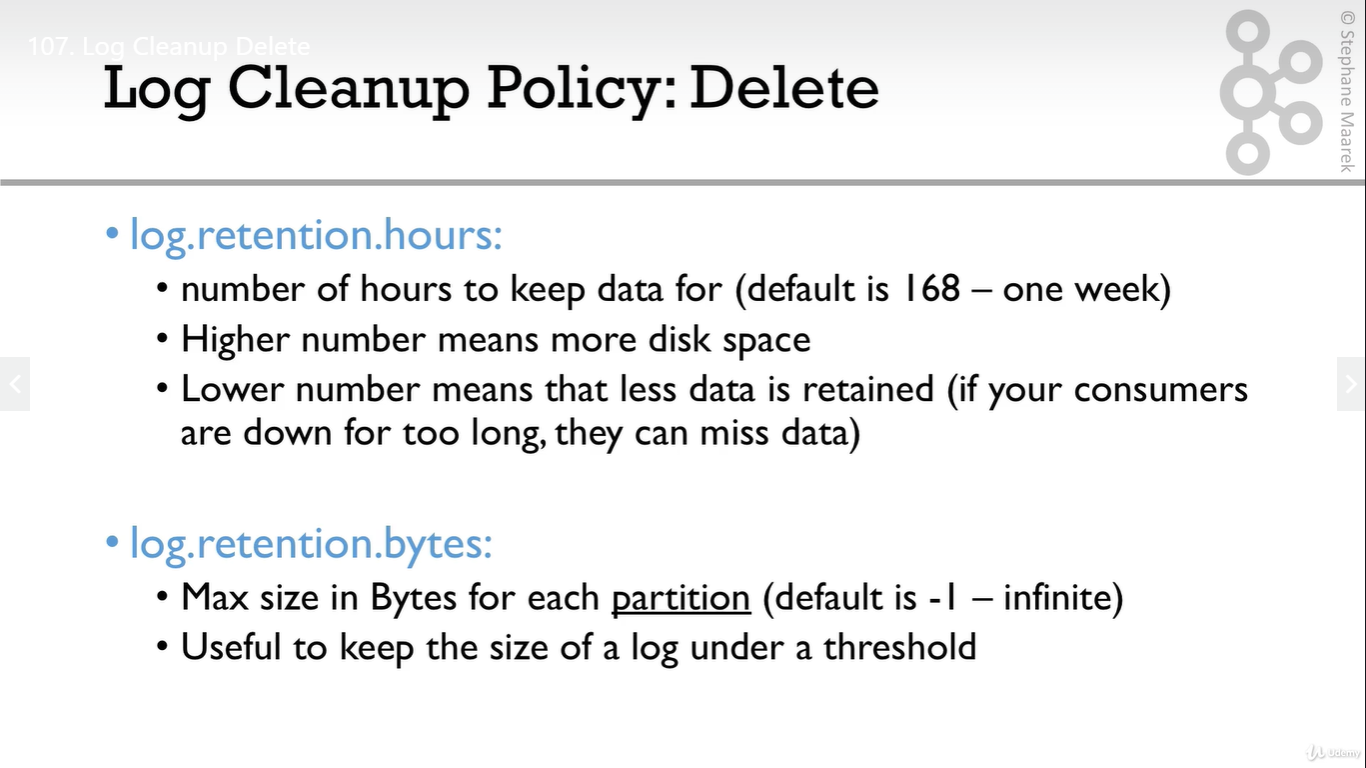


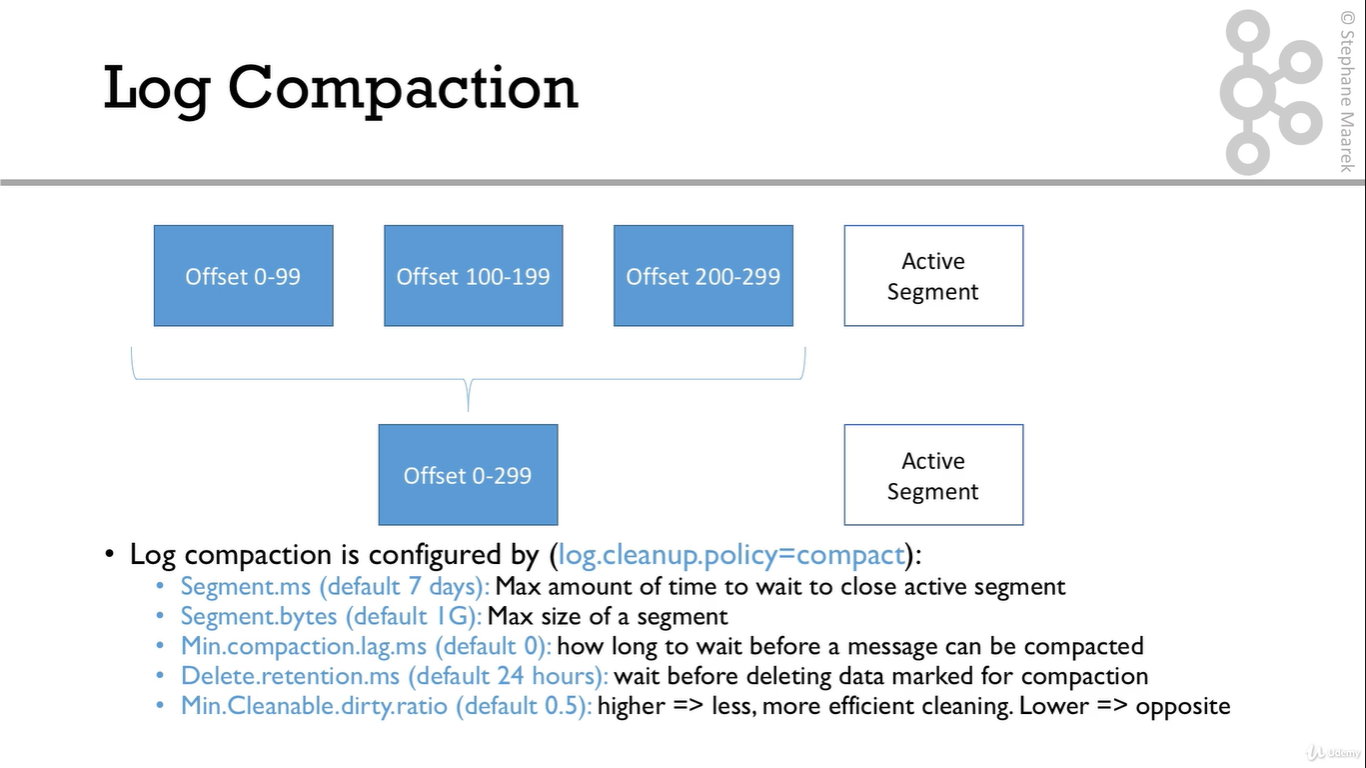












**Kafka.properties: (server.properties in open Hadoop)**

[root@hadoop-slave-1 358-kafka-KAFKA\_BROKER]# head -n 20 kafka.properties

**auto.create.topics.enable=true**

auto.leader.rebalance.enable=true

**broker.id=72**

controlled.shutdown.enable=true

controlled.shutdown.max.retries=3

**default.replication.factor=1**

**delete.topic.enable=true**

kafka.http.metrics.host=0.0.0.0

kafka.http.metrics.port=24042

kafka.log4j.dir=/var/log/kafka

kerberos.auth.enable=false

leader.imbalance.check.interval.seconds=300

leader.imbalance.per.broker.percentage=10

log.cleaner.dedupe.buffer.size=134217728

log.cleaner.delete.retention.ms=604800000

log.cleaner.enable=true

log.cleaner.min.cleanable.ratio=0.5

log.cleaner.threads=1

**log.dirs=/var/local/kafka/data**

log.retention.bytes=-1

**log.retention.check.interval.ms=300000 🡪 every 5 minutes it will check log segment for deletion according to the retention policy**

**log.retention.hours=168 -🡪 7 days**

log.roll.hours=168

**log.segment.bytes=1073741824 -🡪 log segement length eqvals to 1Gb**

message.max.bytes=1000000

**min.insync.replicas=1 --🡪put 2 for safe in production**

num.io.threads=8

**num.partitions=1**

num.replica.fetchers=1

offsets.topic.num.partitions=50

offsets.topic.replication.factor=3

port=9092

replica.fetch.max.bytes=1048576

replica.lag.max.messages=4000

replica.lag.time.max.ms=10000

sentry.kafka.caching.enable=true

sentry.kafka.caching.ttl.ms=30000

sentry.kafka.caching.update.failures.count=3

unclean.leader.election.enable=false

**zookeeper.session.timeout.ms=6000 -🡪 connection timeout 6 seconds**

**zookeeper.connec**t**=hadoop-slave-1.us-central1-a.c.fleet-point-292913.internal:2181,hadoop-slave-2.us-central1-a.c.fleet-point-292913.internal:2181,hadoop-slave-3.us-central1-a.c.fleet-point-292913.internal:2181**

kafka.metrics.reporters=nl.techop.kafka.KafkaHttpMetricsReporter

security.inter.broker.protocol=PLAINTEXT

**listeners=PLAINTEXT://hadoop-slave-1.us-central1-a.c.fleet-point-292913.internal:9092,**

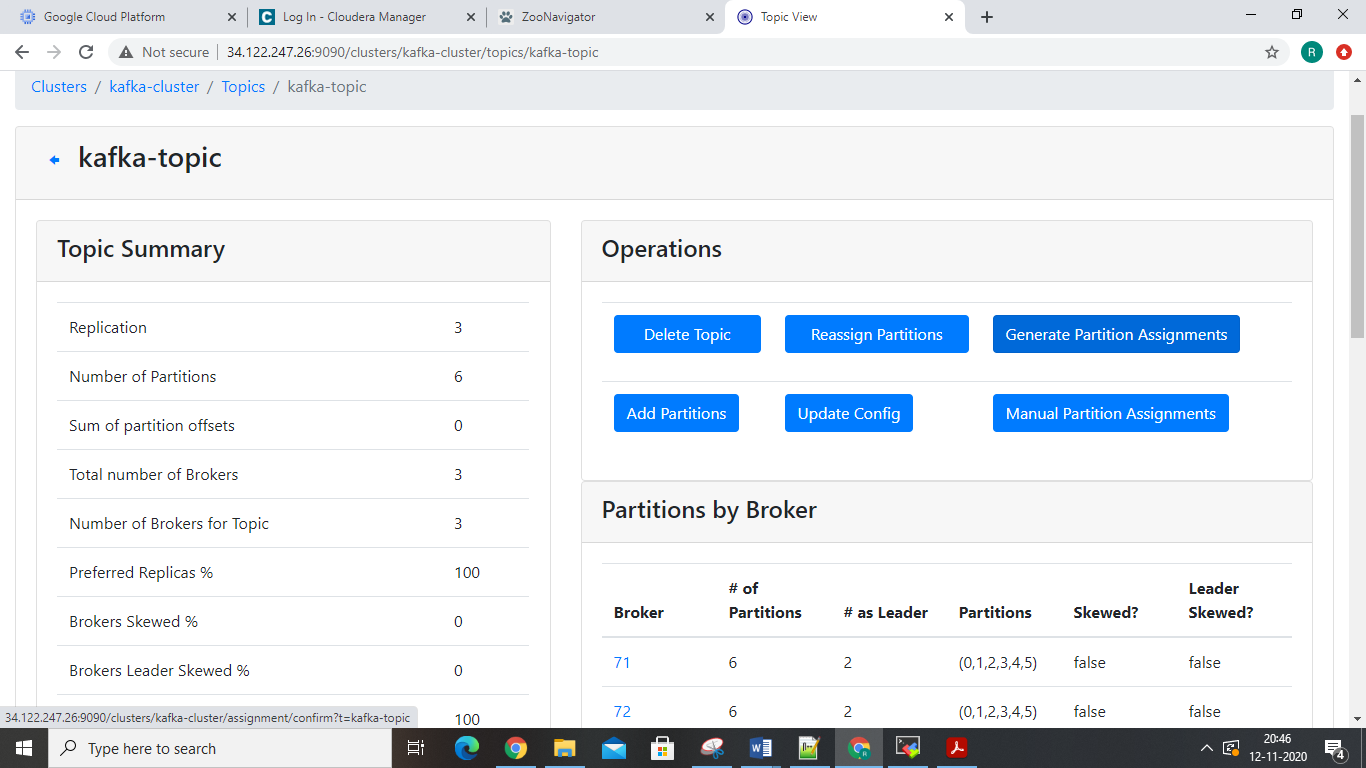
broker.id.generation.enable=false

sasl.kerberos.service.name=kafka

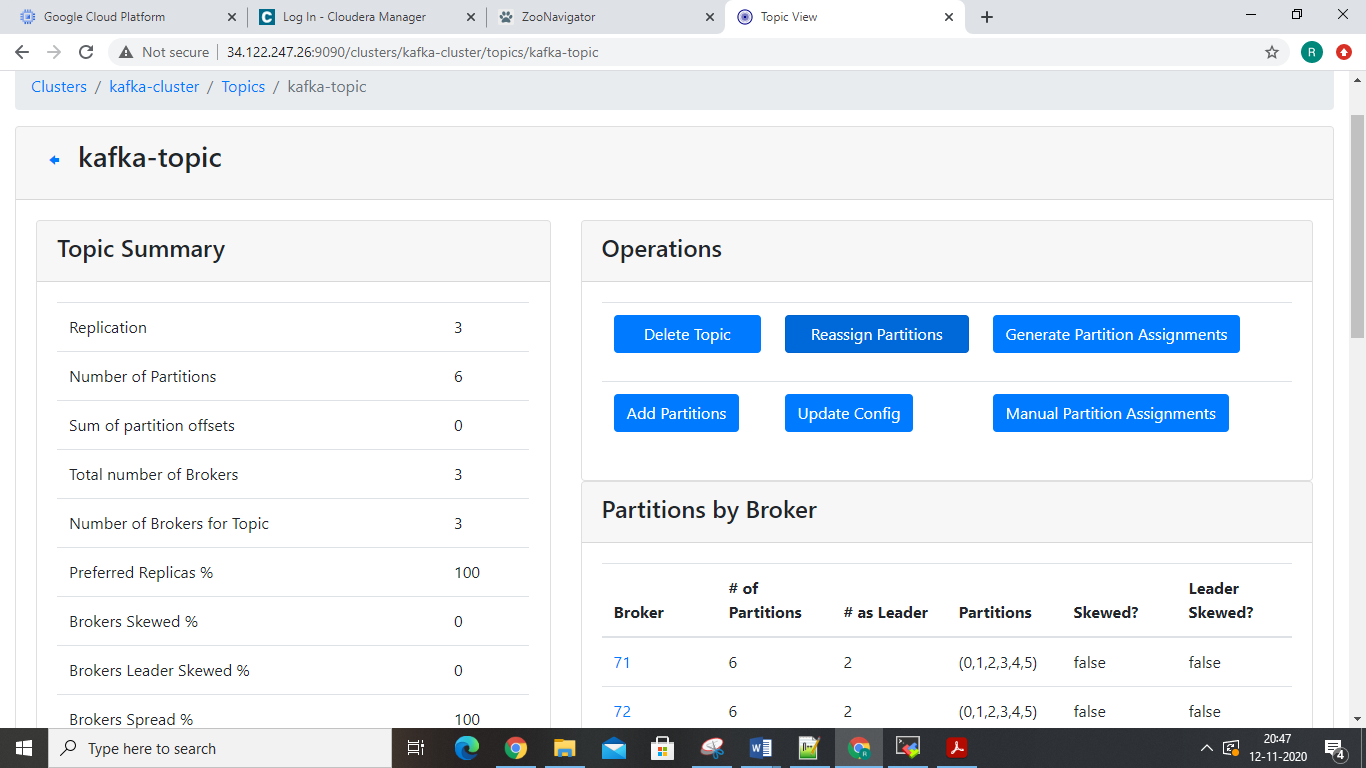
Rebalancing Partitions using Kafka Manager :

Steps:

1. go to topic -🡪Run “Generate partition assignments”.It will generate a Json file for topic reassignment.



2-Then “reassign partitions”.



after reassign Run “Preffered replica election.

Consumer Configuration Properties

There are some very important configurations that any user of Kafka must know:

• **heartbeat.interval.ms**: The interval of the heartbeats. For example, if the heartbeat interval is set to 3

seconds, the consumer sends a short heartbeat message to the broker every 3 seconds to indicate that it is alive.

• **session.timeout.ms**: The consumer tells this timeout to the coordinator. This is used to control the heartbeats

and remove the dead consumers. If it’s set to 10 seconds, the consumer can miss sending 2 heartbeats, assuming

the previous heartbeat setting. If we increase the timeout, the consumer has more room for delays but the broker

notices lagging consumers later.

• **max.poll.interval.ms**: It is a very important detail: the consumers must maintain polling and should never

do long-running processing. If a consumer is taking too much time between two polls, it will be detached from

the consumer group. We can tune this configuration according to our needs. Note that if a consumer is stuck in

processing, it will be noticed later if the value is increased.

• **request.timeout.ms**: Generally every request has a timeout. This is an upper bound that the client waits for

the server’s response. If this timeout elapses, then retries might happen if the number of retries are not exhausted.

Broker Log Management

Kafka brokers save their data as log segments in a directory. The logs are rotated depending on the size and time

settings.

The most common log retention settings to adjust for your cluster are shown below. These are accessible in Cloudera

Manager via the **Kafka** > **Configuration** tab.

• log.dirs: The location for the Kafka data (that is, topic directories and log segments).

• log.retention.{ms|minutes|hours}: The retention period for the entire log. Any older log segments are

removed.

• log.retention.bytes: The retention size for the entire log.

There are many more variables available for fine-tuning broker log management. For more detailed information, look

at the relevant variables in the Apache Kafka documentation topic Broker Configs.

• log.dirs

• log.flush.\*

• log.retention.\*

• log.roll.\*

• log.segment.\*

Record Management

There are two pieces to record management, log segments and log cleaner.

As part of the general data storage, Kafka rolls logs periodically based on size or time limits. Once either limit is hit, a

new log segment is created with the all new data being placed there, while older log segments should generally no

longer change. This helps limit the risk of data loss or corruption to a single segment instead of the entire log.

• log.roll.{ms|hours}: The time period for each log segment. Once the current segment is older than this

value, it goes through log segment rotation.

• log.segment.bytes: The maximum size for a single log segment.

There is an alternative to simply removing log segments for a partition. There is another feature based on the log

cleaner. When the log cleaner is enabled, individual records in older log segments can be managed differently:

• log.cleaner.enable: This is a global setting in Kafka to enable the log cleaner.

• cleanup.policy: This is a per-topic property that is usually set at topic creation time. There are two valid values

for this property, delete and compact.

• log.cleaner.min.compaction.lag.ms: This is the retention period for the “head” of the log. Only records

outside of this retention period will be compacted by the log cleaner.

The compact policy, also called log compaction, assumes that the "most recent Kafka record is important." Some

examples include tracking a current email address or tracking a current mailing address. With log compaction, older

records with the same key are removed from a log segment and the latest one is kept. This effectively removes some

offsets from the partition.

**60** | Apache Kafka Guide

Kafka Administration

Broker Garbage Log Collection and Log Rotation

Both broker JVM garbage collection and JVM garbage log rotation is enabled by default in the Kafka version delivered

with CDH. Garbage collection logs are written in the agent process directory by default.

Example path:

/run/cloudera-scm-agent/process/99-kafka-KAFKA\_BROKER/kafkaServer-gc.log

Changing the default directory of garbage collection logs is currently not supported. However, you can configure

properties related garbage log rotation with the **Kafka Broker Environment Advanced Configuration Snippet (Safety**

**Valve)** property.

**1.** In Cloudera Manager, go to the Kafka service and click **Configuration**.

**2.** Find the **Kafka Broker Environment Advanced Configuration Snippet (Safety Valve)** property.

**3.** Add the following line to the property:

Modify the values of as required.

KAFKA\_GC\_LOG\_OPTS="-XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=*10*

-XX:GCLogFileSize=*100M*"

The flags used are as follows:

• +UseGCLogFileRotation: Enables garbage log rotation.

• -XX:NumberOfGCLogFiles: Specifies the number of files to use when rotating logs.

• -XX:GCLogFileSize: Specifies the size when the log will be rotated.

**4.** Click on **Save Changes**.

**5.** Restart the Kafka service to apply the changes.

Adding Users as Kafka Administrators

In some cases, additional users besides the kafka account need administrator access. This can be done in Cloudera

Manager by going to **Kafka** > **Configuration** > **Super users**.

Migrating Brokers in a Cluster

Brokers can be moved to a new host in a Kafka cluster. This might be needed in the case of catastrophic hardware

failure. Make sure the following are true before starting:

• Make sure the cluster is healthy.

• Make sure all replicas are in sync.

• Perform the migration when there is minimal load on the cluster.

Brokers need to be moved one-by-one. There are two techniques available:

**Using kafka-reassign-partitions tool**

This method involves more manual work to modify JSON, but does not require manual edits to configuration files.

For more information, see kafka-reassign-partitions on page 66.

**Modify the broker IDs in meta.properties**

This technique involves less manual work, but requires modifying an internal configuration file.

**1.** Start up the new broker as a member of the old cluster.

This creates files in the data directory.

**2.** Stop both the new broker and the old broker that it is replacing.

Apache Kafka Guide | **61**

Kafka Administration

**3.** Change broker.id of the new broker to the broker.id of the old one both in Cloudera Manager and in

*data directory*/meta.properties.

**4.** (Optional) Run rsync to copy files from one broker to another.

See Using rsync to Copy Files from One Broker to Another on page 62.

**5.** Start up the new broker.

It re-replicates data from the other nodes.

Note that data intensive administration operations such as rebalancing partitions, adding a broker, removing a broker,

or bootstrapping a new machine can cause significant additional load on the cluster.

To avoid performance degradation of business workloads, you can limit the resources that these background processes

can consume by specifying the -throttleparameter when running kafka-reassign-partitions.

Using rsync to Copy Files from One Broker to Another

You can run rsync command to copy over all data from an old broker to a new broker, preserving modification times

and permissions. Using rsync allows you to avoid having to re-replicate the data from the leader. You have to ensure

that the disk structures match between the two brokers, or you have to verify the meta.properties file between

the source and destination brokers (because there is one meta.properties file for each data directory).

Run the following command on destination broker:

rsync -avz

*src\_broker*:*src\_data\_dir*

*dest\_data\_dir*

If you plan to change the broker ID, edit *dest\_data\_dir*/meta.properties.

Setting User Limits for Kafka

Kafka opens many files at the same time. The default setting of 1024 for the maximum number of open files on most

Unix-like systems is insufficient. Any significant load can result in failures and cause error messages such as

java.io.IOException...(Too many open files) to be logged in the Kafka or HDFS log files. You might also

notice errors such as this:

ERROR Error in acceptor (kafka.network.Acceptor)

java.io.IOException: Too many open files

Cloudera recommends setting the value to a relatively high starting point, such as 32,768.

You can monitor the number of file descriptors in use on the Kafka Broker dashboard. In Cloudera Manager:

**1.** Go to the Kafka service.

**2.** Select a Kafka Broker.

**3.** Open **Charts Library** > **Process Resources** and scroll down to the **File Descriptors** chart.

See Viewing Charts for Cluster, Service, Role, and Host Instances.

Quotas

For a quick video introduction to quotas, see Quotas.

In CDK 2.0 Powered by Apache Kafka and higher, Kafka can enforce quotas on produce and fetch requests. Producers

and consumers can use very high volumes of data. This can monopolize broker resources, cause network saturation,

and generally deny service to other clients and the brokers themselves. *Quotas* protect against these issues and are

important for large, multi-tenant clusters where a small set of clients using high volumes of data can degrade the user

experience.

**62** | Apache Kafka Guide

Kafka Administration

Quotas are byte-rate thresholds, defined per client ID. A client ID logically identifies an application making a request.

A single client ID can span multiple producer and consumer instances. The quota is applied for all instances as a single

entity. For example, if a client ID has a produce quota of 10 MB/s, that quota is shared across all instances with that

same ID.

When running Kafka as a service, quotas can enforce API limits. By default, each unique client ID receives a fixed quota

in bytes per second, as configured by the cluster (quota.producer.default, quota.consumer.default). This

quota is defined on a per-broker basis. Each client can publish or fetch a maximum of *X* bytes per second per broker

before it gets throttled.

The broker does not return an error when a client exceeds its quota, but instead attempts to slow the client down.

The broker computes the amount of delay needed to bring a client under its quota and delays the response for that

amount of time. This approach keeps the quota violation transparent to clients (outside of client-side metrics). This

also prevents clients from having to implement special backoff and retry behavior.

Setting Quotas

You can override the default quota for client IDs that need a higher or lower quota. The mechanism is similar to per-topic

log configuration overrides. Write your client ID overrides to ZooKeeper under /config/clients. All brokers read

the overrides, which are effective immediately. You can change quotas without having to do a rolling restart of the

entire cluster.

By default, each client ID receives an unlimited quota. The following configuration sets the default quota per producer

and consumer client ID to 10 MB/s.

quota.producer.default=10485760

quota.consumer.default=10485760

Kafka Performance Tuning

Performance tuning involves two important metrics:

• Latency measures how long it takes to process one event.

• Throughput measures how many events arrive within a specific amount of time.

Most systems are optimized for either latency or throughput. Kafka is balanced for both. A well-tuned Kafka system

has just enough brokers to handle topic throughput, given the latency required to process information as it is received.

Tuning your producers, brokers, and consumers to send, process, and receive the largest possible batches within a

manageable amount of time results in the best balance of latency and throughput for your Kafka cluster.

The following sections introduce the concepts you'll need to be able to balance your Kafka workload and then provide

practical tuning configuration to address specific circumstances.

For a quick video introduction to tuning Kafka, see Tuning Your Apache Kafka Cluster.

There are a few concepts described here that will help you focus your tuning efforts. Additional topics in this section

provide practical tuning guidelines:

Tuning Brokers

Topics are divided into partitions. Each partition has a leader. Topics that are properly configured for reliability will

consist of a leader partition and 2 or more follower partitions. When the leaders are not balanced properly, one might

be overworked, compared to others.

Depending on your system and how critical your data is, you want to be sure that you have sufficient replication sets

to preserve your data. For each topic, Cloudera recommends starting with one partition per physical storage disk and

one consumer per partition.

Tuning Producers

Kafka uses an asynchronous publish/subscribe model. When your producer calls send(), the result returned is a future.

The future provides methods to let you check the status of the information in process. When the batch is ready, the

producer sends it to the broker. The Kafka broker waits for an event, receives the result, and then responds that the

transaction is complete.

If you do not use a future, you could get just one record, wait for the result, and then send a response. Latency is very

low, but so is throughput. If each transaction takes 5 ms, throughput is 200 events per second — slower than the

expected 100,000 events per second.

When you use Producer.send(), you fill up buffers on the producer. When a buffer is full, the producer sends the

buffer to the Kafka broker and begins to refill the buffer.

Two parameters are particularly important for latency and throughput: batch size and linger time.

Batch Size

batch.size measures batch size in total bytes instead of the number of messages. It controls how many bytes of

data to collect before sending messages to the Kafka broker. Set this as high as possible, without exceeding available

memory. The default value is 16384.

If you increase the size of your buffer, it might never get full. The Producer sends the information eventually, based

on other triggers, such as linger time in milliseconds. Although you can impair memory usage by setting the buffer

batch size too high, this does not impact latency.

**86** | Apache Kafka Guide

Kafka Performance Tuning

If your producer is sending all the time, you are probably getting the best throughput possible. If the producer is often

idle, you might not be writing enough data to warrant the current allocation of resources.

Linger Time

linger.ms sets the maximum time to buffer data in asynchronous mode. For example, the setting of 100 means that

it batches 100ms of messages to send at once. This improves throughput, but the buffering adds message delivery

latency.

By default, the producer does not wait. It sends the buffer any time data is available.

Instead of sending immediately, you can set linger.ms to 5 and send more messages in one batch. This would reduce

the number of requests sent, but would add up to 5 milliseconds of latency to records sent, even if the load on the

system does not warrant the delay.

The farther away the broker is from the producer, the more overhead required to send messages. Increase linger.ms

for higher latency and higher throughput in your producer.

Tuning Consumers

Consumers can create throughput issues on the other side of the pipeline. The maximum number of consumers in a

consumer group for a topic is equal to the number of partitions. You need enough partitions to handle all the consumers

needed to keep up with the producers.

Consumers in the same consumer group split the partitions among them. Adding more consumers to a group can

enhance performance (up to the number of partitions). Adding more consumer groups does not affect performance.

# Q: [How to prevent “Write Failed: broken pipe” on SSH connection?](https://askubuntu.com/questions/127369/how-to-prevent-write-failed-broken-pipe-on-ssh-connection)

# What can I do to configure SSH on both client and servers to prevent Write Failed: broken pipe errors? It often occurs if you sleep your client computer and resume later.

I have tried this in /etc/ssh/ssh\_config for Linux and Mac:

Host \*

ServerAliveInterval 120

This is how often, in seconds, it should send a keepalive message to the server. If that doesn't work then train a monkey to press enter every two minutes while you work.

You could set either ServerAliveInterval in /etc/ssh/ssh\_config of the client machine or ClientAliveInterval in /etc/ssh/sshd\_config of the server machine. Try reducing the interval if you are still getting the error.

Configuration for a single user can be set in file ~/.ssh/config both on the server and client side. Make sure the file has correct permissions chmod 644 ~/.ssh/config.

# Q: what are the depenedncy of ansible ...or prerequisite ?

# Ans: ans: passwrd less connection and python module.

# Q: if you are not able to see table table in impala,then what command will use ?

# Ans: invalidate meatadata

**How do you debug a performance issue or a long running job in Hadoop?**

**This post will explain how can you approach the above question when asked in an interview.**

This is an open ended interview question and the interviewer is trying to see the level of hands-on experience you have in solving production issues. Use your day to day work experience to answer this question. Here are some of the scenarios and responses to help you construct your answer. On a very high level you will follow the below steps.

* Understand the symptom
* Analyze the situation
* Identify the problem areas
* Propose solution

Scenario 1 – Job with 100 mappers and 1 reducer takes a long time for the reducer to start after all the mappers are complete. One of the reasons could be that reduce is spending a lot of time copying the map outputs. So in this case we can try couple of things.

1. If possible add a combiner to reduce the amount of output from the mapper to be sent to the reducer
2. Enable map output compression – this will further reduce the size of the outputs to be transferred to the reducer.

Scenario 2 – A particular task is using a lot of memory which is causing the slowness or failure, I will look for ways to reduce the memory usage.

1. Make sure the joins are made in an optimal way with memory usage in mind. For e.g. in Pig joins, the LEFT hand side tables are sent to the reducer first and held in memory and the RIGHT most table in streamed to the reducer. So make sure the RIGHT most table is largest of the datasets in the join.
2. We can also increase the memory requirements needed by the map and reduce tasks by setting – *mapreduce.map.memory.mb* and *mapreduce.reduce.memory.mb*

Scenario 3 – Understanding the data helps a lot in optimizing the way we use the datasets in PIG and HIVE scripts.

1. If you have smaller tables in join, they can be sent to distributed cache and loaded in memory on the Map side and the entire join can be done on the Map side thereby avoiding the shuffle and reduce phase altogether. This will tremendously improve performance. Look up *USING REPLICATED* in Pig and *MAPJOIN* or *hive.auto.convert.join* in Hive
2. If the data is already sorted you can use *USING MERGE* which will do a Map Only join
3. If the data is bucketted in hive, you may use *hive.optimize.bucketmapjoin* or *hive.optimize.bucketmapjoin.sortedmerge* depending on the characteristics of the data

Scenario 4 – The Shuffle process is the heart of a MapReduce program and it can be tweaked for performance improvement.

1. If you see lots of records are being spilled to the disk (check for Spilled Records in the counters in your MapReduce output) you can increase the memory available for Map to perform the Shuffle by increasing the value in *io.sort.mb*. This will reduce the amount of Map Outputs written to the disk so the sorting of the keys can be performed in memory.
2. On the reduce side the merge operation (merging the output from several mappers) can be done in disk by setting the *mapred.inmem.merge.threshold* to 0