Hortonworks Data Platform

System Administration Guides

(Oct 22, 2013)

docs.hortonworks.com

Hortonworks Data Platform: System Administration Guides

The Hortonworks Data Platform, powered by Apache Hadoop, is a massively scalable and 100% open source platform for storing, processing and analyzing large volumes of data. It is designed to deal with data from many sources and formats in a very quick, easy and cost-effective manner. The Hortonworks Data Platform consists of the essential set of Apache Hadoop projects including MapReduce, Hadoop Distributed File System (HDFS), HCatalog, Pig, Hive, HBase, Zookeeper and Ambari. Hortonworks is the major contributor of code and patches to many of these projects. These projects have been integrated and tested as part of the Hortonworks Data Platform release process and installation and configuration tools have also been included.

Unlike other providers of platforms built using Apache Hadoop, Hortonworks contributes 100% of our code back to the Apache Software Foundation. The Hortonworks Data Platform is Apache-licensed and completely open source. We sell only expert technical support, training and partner-enablement services. All of our technology is, and will remain free and open source.

Please visit the Hortonworks Data Platform page for more information on Hortonworks technology. For more information on Hortonworks services, please visit either the Support or Training page. Feel free to Contact Us directly to discuss your specific needs.

Table of Contents

1. High Availability for Hadoop

This guide provides an overview of the HDFS High Availability (HA) feature and instructions on configuring and managing a highly available (HA) HDFS cluster using the Quorum Journal Manager (QJM) feature and Zookeeper Failover Controller.

This guide provides instructions on configuring and using HDFS HA using the Quorum Journal Manager (QJM) and the Zookeeper Failover Controller in order to share edit logs between the Active and Standby NameNodes.

Note

This guide assumes that an existing HDP cluster has been installed manually and deployed. It provides instructions on how to enable HA on top of the existing cluster. If Ambari was used to install that existing cluster, refer to the Ambari documentation for more details on configuring NameNode HA using the Ambari wizard.

The NameNode was a single point of failure (SPOF) in an HDFS cluster. Each cluster had a single NameNode and if that machine or process became unavailable, entire cluster would be unavailable until the NameNode was either restarted or started on a separate machine. This situation impacted the total availability of the HDFS cluster in two major ways:

- In the case of an unplanned event such as a machine crash, the cluster would be unavailable until an operator restarted the NameNode.
- Planned maintenance events such as software or hardware upgrades on the NameNode machine would result in windows of cluster downtime.

The HDFS HA feature addresses these problems. The HA feature lets you run redundant NameNodes in the same cluster in an Active/Passive configuration with a hot standby. This mechanism thus facilitates either a fast failover to the new NameNode during machine crash or a graceful administrator-initiated failover during planned maintenance.

In this document:

1.1. Architecture

In a typical HA cluster, two separate machines are configured as NameNodes. In a working cluster, one of the NameNode machine is in the **Active** state, and the other is in the **Standby** state.

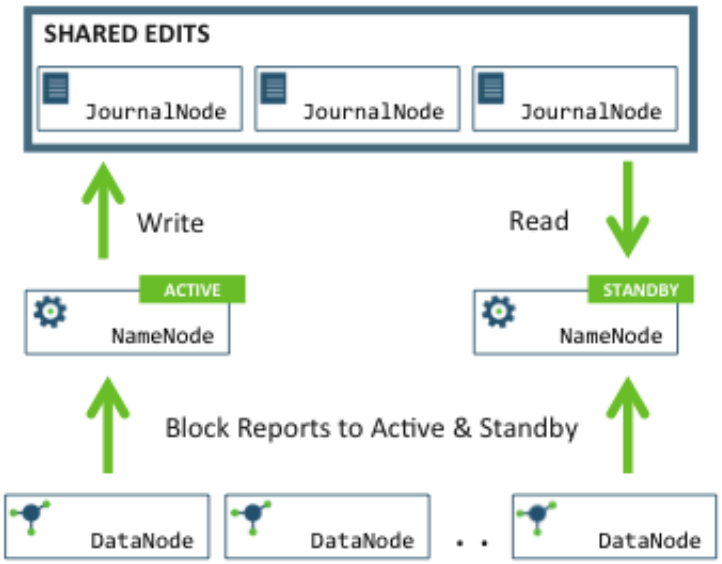
The Active NameNode is responsible for all client operations in the cluster, while the Standby is acts as a slave. The Standby machine maintains enough state to provide a fast failover (if required).

In order for the Standby node to keep its state synchronized with the Active node, both nodes communicate with a group of separate daemons called **JournalNodes** (JNs). When the Active node performs any namespace modification, the Active node durably logs a modification record to a majority of these JNs. The Standby node reads the edits from the JNs and continuously watches the JNs for changes to the edit log. Once the Standby Node observes the edits, it applies these edits to its own namespace. When using QJM, JournalNodes acts the shared editlog storage. In a failover event, the Standby ensures that it has read all of the edits from the JounalNodes before promoting itself to the Active state. (This mechanism ensures that the namespace state is fully synchronized before a failover completes.)

Note

Secondary NameNode is not required in HA configuration because the Standby node also performs the tasks of the Secondary NameNode.

In order to provide a fast failover, it is also necessary that the Standby node have up-to-date information of the location of blocks in your cluster. To get accurate information of the block locations, DataNodes are configured with the location of both the NameNodes and send block location information and heartbeats to both NameNode machines.



It is vital for the correct operation of an HA cluster that only one of the NameNodes should be Active at a time. Failure to do so, would cause the namespace state to quickly diverge between the two NameNode machines thus causing potential data loss. (This situation is called as **split-brain scenario**.)

To prevent the **split-brain scenario**, the JournalNodes allow only one NameNode to be a writer at a time. During failover, the NameNode, that is to chosen to become active, takes over the role of writing to the JournalNodes. This process prevents the other NameNode

from continuing in the Active state and thus lets the new Active node proceed with the failover safely.

1.2. Hardware Resources

Ensure that you prepare the following hardware resources:

- **NameNode machines**: The machines where you run Active and Standby NameNodes, should have exact same hardware. For recommended hardware for Hadoop, see Hardware recommendations for Apache Hadoop.
- **JournalNode machines**: The machines where you run the JournalNodes. The JournalNode daemon is relatively lightweight, so these daemons may reasonably be co-located on machines with other Hadoop daemons, for example NameNodes, the JobTracker, or the YARN ResourceManager.

Note

There must be at least three JournalNode daemons, because edit log modifications must be written to a majority of JNs. This lets the system tolerate failure of a single machine. You may also run more than three JournalNodes, but in order to increase the number of failures that the system can tolerate, you must run an odd number of JNs, (i.e. 3, 5, 7, etc.).

Note that when running with `N` JournalNodes, the system can tolerate at most `(N - 1) / 2` failures and continue to function normally.

- **Zookeeper machines**: For automated faolover functionality, there must be an existing Zookeeper cluster available. The Zookeeper service nodes can be co-located with other Hadoop daemons.

In an HA cluster, the Standby NameNode also performs checkpoints of the namespace state, and therefore do not deploy a Secondary NameNode, CheckpointNode, or BackupNode in an HA cluster.

1.3. Deploy HA Cluster

HA configuration is backward compatible and works with your existing single NameNode configuration.

Note

These instructions provide information on setting up NameNode HA on a manually installed cluster. If you installed with Ambari and manage HDP 2.0 on Ambari 1.4.1 or later, use the Ambari documentation for the NameNode HA wizard instead of these instructions.

First, enable the NameNodes to share metadata through QJM. Then, enable automatic failover through Zookeeper.

- Configure HA cluster
- Deploy manual failover

1.3.1. Configure HA Cluster

Add High Availability configurations to your HDFS configuration files. Start by taking the HDFS configuration files from the original NameNode in your HDP cluster, and use that as the base, adding the properties mentioned below to those files.

After you have added the configurations below, ensure that the same set of HDFS configuration files are propogated to all nodes in the HDP cluster. This ensures that all the nodes and services are able to interact with the highly available NameNode.

Add the following configuration options to your `hdfs-site.xml` file:

- **dfs.nameservices:**

  Choose an arbitrary but logical name (for example `mycluster`) as the value for `dfs.nameservices` option. This name will be used for both configuration and authority component of absolute HDFS paths in the cluster.

```
<property>
  <name>dfs.nameservices</name>
  <value>mycluster</value>
  <description>Logical name for this new nameservice</description>
</property>
```

  If you are also using HDFS Federation, this configuration setting should also include the list of other nameservices, HA or otherwise, as a comma-separated list.

- **dfs.ha.namenodes.[$nameservice ID]:**

  Provide a list of comma-separated NameNode IDs. DataNodes use this this property to determine all the NameNodes in the cluster.

  For example, for the nameservice ID `mycluster` and individual NameNode IDs `nn1` and `nn2`, the value of this property will be:

```
<property>
  <name>dfs.ha.namenodes.mycluster</name>
  <value>nn1,nn2</value>
  <description>Unique identifiers for each NameNode in the nameservice</
description>
</property>
```

Note

Currently, a maximum of two NameNodes may be configured per nameservice.

- **dfs.namenode.rpc-address.[$nameservice ID].[$name node ID]:**

  Use this property to specify the fully-qualified RPC address for each NameNode to listen on.

  Continuning with the previous example, set the full address and IPC port of the NameNode process for the above two NameNode IDs - `nn1` and `nn2` .

Note that there will be two separate configuration options.

```
<property>
  <name>dfs.namenode.rpc-address.mycluster.nn1</name>
  <value>machine1.example.com:8020</value>
</property>
<property>
  <name>dfs.namenode.rpc-address.mycluster.nn2</name>
  <value>machine2.example.com:8020</value>
</property>
```

- **dfs.namenode.http-address.[$nameservice ID].[$name node ID]:**

  Use this property to specify the fully-qualified HTTP address for each NameNode to listen on.

  Set the addresses for both NameNodes HTTP servers to listen on. For example:

```
<property>
<name>dfs.namenode.http-address.mycluster.nn1</name>
<value>machine1.example.com:50070</value>
</property>
<property>
<name>dfs.namenode.http-address.mycluster.nn2</name>
<value>machine2.example.com:50070</value>
</property>
```

  Note

  If you have Hadoop security features enabled, set the https-address for each NameNode.

- **dfs.namenode.shared.edits.dir:**

  Use this property to specify the URI that identifies a group of JournalNodes (JNs) where the NameNode will write/read edits.

  Configure the addresses of the JNs that provide the shared edits storage. The Active nameNode writes to this shared storage and the Standby NameNode reads from this location to stay up-to-date with all the file system changes.

  Although you must specify several JournalNode addresses, **you must configure only one of these URIs** for your cluster.

  The URI should be of the form:

```
qjournal://host1:port1;host2:port2;host3:port3/journalId
```

  The Journal ID is a unique identifier for this nameservice, which allows a single set of JournalNodes to provide storage for multiple federated namesystems. You can reuse the nameservice ID for the journal identifier.

  For example, if the JournalNodes for a cluster were running on the `node1.example.com`, `node2.example.com`, and `node3.example.com` machines

and the nameservice ID were `mycluster`, you would use the following as the value for this setting:

```
<property>
  <name>dfs.namenode.shared.edits.dir</name>
  <value>qjournal://node1.example.com:8485;node2.example.com:8485;node3.
example.com:8485/mycluster</value>
</property>
```

Note

Note that the default port for the JournalNode is 8485.

- **dfs.client.failover.proxy.provider.[$nameservice ID]:**

  This property specifies the Java class that HDFS clients use to contact the Active NameNode. DFS Client uses this Java class to determine which NameNode is the current Active and therefore which NameNode is currently serving client requests.

  Use the **ConfiguredFailoverProxyProvider** implementation if you are not using a custom implementation.

  For example:

```
<property>
  <name>dfs.client.failover.proxy.provider.mycluster</name>
  <value>org.apache.hadoop.hdfs.server.namenode.ha.
ConfiguredFailoverProxyProvider</value>
</property>
```

- **dfs.ha.fencing.methods:**

  This property specifies a list of scripts or Java classes that will be used to fence the Active NameNode during a failover.

  It is important for maintaining correctness of the system that only one NameNode be in the Active state at any given time. Especially, when using the Quorum Journal Manager, only one NameNode will ever be allowed to write to the JournalNodes, so there is no potential for corrupting the file system metadata from a split-brain scenario. However, when a failover occurs, it is still possible that the previous Active NameNode could serve read requests to clients, which may be out of date until that NameNode shuts down when trying to write to the JournalNodes. For this reason, it is still recommended to configure some fencing methods even when using the Quorum Journal Manager. To improve the availability of the system in the event the fencing mechanisms fail, it is advisable to configure a fencing method which is guaranteed to return success as the last fencing method in the list. Note that if you choose to use no actual fencing methods, you must set some value for this setting, for example `shell(/bin/true)`.

  The fencing methods used during a failover are configured as a carriage-return-separated list, which will be attempted in order until one indicates that fencing has succeeded. The following two methods are packaged with Hadoop: `shell` and `sshfence`. For information on implementing custom fencing method, see the `org.apache.hadoop.ha.NodeFencer` class.

  - **sshfence:** SSH to the Active NameNode and kill the process.

The *sshfence* option SSHes to the target node and uses *fuser* to kill the process listening on the service's TCP port. In order for this fencing option to work, it must be able to SSH to the target node without providing a passphrase. Ensure that you configure the **dfs.ha.fencing.ssh.private-key-files** option, which is a comma-separated list of SSH private key files.

For example:

```
<property>
  <name>dfs.ha.fencing.methods</name>
  <value>sshfence</value>
</property>

<property>
  <name>dfs.ha.fencing.ssh.private-key-files</name>
  <value>/home/exampleuser/.ssh/id_rsa</value>
</property>
```

Optionally, you can also configure a non-standard username or port to perform the SSH. You can also configure a timeout, in milliseconds, for the SSH, after which this fencing method will be considered to have failed. To configure non-standard username or port and timeout, see the properties given below:

```
<property>
  <name>dfs.ha.fencing.methods</name>
  <value>sshfence([[username][:port]])</value>
</property>
<property>
  <name>dfs.ha.fencing.ssh.connect-timeout</name>
  <value>30000</value>
</property>
```

- **shell:** Run an arbitrary shell command to fence the Active NameNode.

  The *shell* fencing method runs an arbitrary shell command:

  ```
  <property>
    <name>dfs.ha.fencing.methods</name>
    <value>shell(/path/to/my/script.sh arg1 arg2 ...)</value>
  </property>
  ```

  The string between '(' and ')' is passed directly to a bash shell and may not include any closing parentheses.

  The shell command will be run with an environment set up to contain all of the current Hadoop configuration variables, with the '_' character replacing any '.' characters in the configuration keys. The configuration used has already had any namenode-specific configurations promoted to their generic forms – for example **dfs_namenode_rpc-address** will contain the RPC address of the target node, even though the configuration may specify that variable as **dfs.namenode.rpc-address.ns1.nn1**.

  Additionally, the following variables (referring to the target node to be fenced) are also available:

- *$target_host*: Hostname of the node to be fenced
- *$target_port*: IPC port of the node to be fenced
- *$target_address*: The combination of *$target_host* and *$target_port* as host:port
- *$target_nameserviceid*: The nameservice ID of the NN to be fenced
- *$target_namenodeid*: The namenode ID of the NN to be fenced

These environment variables may also be used as substitutions in the shell command. For example:

```
<property>
  <name>dfs.ha.fencing.methods</name>
  <value>shell(/path/to/my/script.sh --nameservice=$target_nameserviceid
 $target_host:$target_port)</value>
</property>
```

If the shell command returns an exit code of 0, the fencing is successful.

Note

This fencing method does not implement any timeout. If timeouts are necessary, they should be implemented in the shell script itself (for example, by forking a subshell to kill its parent in some number of seconds).

- **fs.defaultFS:**

  The default path prefix used by the Hadoop FS client. Optionally, you may now configure the default path for Hadoop clients to use the new HA-enabled logical URI. For example, for `mycluster` nameservice ID, this will be the value of the authority portion of all of your HDFS paths.

  Configure this property in the **core-site.xml** file:

```
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://mycluster</value>
</property>
```

- **dfs.journalnode.edits.dir:**

  This is the absolute path on the JournalNode machines where the edits and other local state (used by the JNs) will be stored. You may only use a single path for this configuration. Redundancy for this data is provided by either running multiple separate JournalNodes or by configuring this directory on a locally-attached RAID array. For example:

```
<property>
  <name>dfs.journalnode.edits.dir</name>
  <value>/path/to/journal/node/local/data</value>
</property>
```

1.3.2. Deploy HA Cluster

This document uses the following conventions:

- **NN1** is used to denote the original NameNode in the non-HA setup
- **NN2** is used to denote the other NameNode that is to be added in the HA setup.

Note

HA clusters reuse the `nameservice ID` to identify a single HDFS instance (that may consist of multiple HA NameNodes).

A new abstraction called `NameNode ID` is added with HA. Each NameNode in the cluster has a distinct `NameNode ID` to distinguish it.

To support a single configuration file for all of the NameNodes, the relevant configuration parameters are suffixed with both the **nameservice ID** and the **NameNode ID**.

1. Start the JournalNode daemons on those set of machines where the JNs are deployed. On the **NN1** host machine, execute the following command:

   ```
   hdfs-daemon.sh start journalnode
   ```

   Wait for the daemon to start on each of the JN machines.

   NN1 is the original NameNode machine in your non-HA cluster.

2. Stop NN1. Execute the following command on the NN1 host machine as the HDFS user:

   ```
   su -l hdfs -c "/usr/lib/hadoop/sbin/hadoop-daemon.sh --config /etc/hadoop/
   conf stop namenode"
   ```

3. Initialize JournalNodes.

   Execute the following command on the **NN1** host machine:

   ```
   hdfs namenode -initializeSharedEdits [-force | -nonInteractive]
   ```

   This command performs the following tasks:

   - Formats all the JournalNodes.

     This by default happens in an interactive way: the command prompts users for “Y/N” input to confirm the format.

     You can skip the prompt by using option **-force** or **-nonInteractive**.

   - Copies all the edits data after the most recent checkpoint from the edits directories of the local NameNode (**NN1**) to JournalNodes.

4. Start **NN1**. Execute the following command on the NN1 host machine as the HDFS user:

   ```
   su -l hdfs -c "/usr/lib/hadoop/sbin/hadoop-daemon.sh --config /etc/hadoop/
   conf start namenode"
   ```

Ensure that NN1 is running correctly.

5. Initialize NN2.

   Format NN2 and copy the latest checkpoint (FSImage) from NN1 to NN2 by executing the following command:

   ```
   hdfs namenode -bootstrapStandby [-force | -nonInteractive]
   ```

   This command connects with HH1 to get the namespace metadata and the checkpointed fsimage. This command also ensures that NN2 receives sufficient editlogs from the JournalNodes (corresponding to the fsimage). This command fails if JournalNodes are not correctly initialized and cannot provide the required editlogs.

6. Start NN2. Execute the following command on the NN2 host machine:

   ```
   /usr/lib/hadoop/sbin/hadoop-daemon.sh start namenode
   ```

   Ensure that NN2 is running correctly.

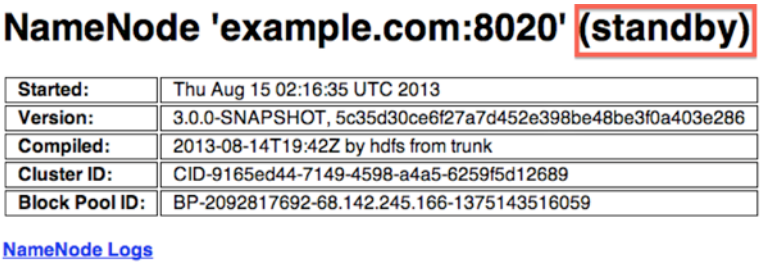
7. Start DataNodes. Execute the following command on all the DataNodes:

   ```
   su -l hdfs -c "/usr/lib/hadoop/sbin/hadoop-daemon.sh --config /etc/hadoop/
   conf start datanode"
   ```

8. Validate HA configuration.

   Go to the NameNodes' web pages separately by browsing to their configured HTTP addresses.

   Under the configured address label, you should see that HA state of the NameNode. The NameNode can be either in "standby" or "active" state.

   Note

   The HA NameNode is initially in the Standby state after it is bootstrapped.

   You can also use either JMX (`tag.HAState`) to query the HA state of a NameNode.

   The following command can also be used query HA state for NameNode:

   ```
   hdfs haadmin -getServiceState
   ```

9. Transition one of the HA NameNode to Active state.

Initially, both NN1 and NN2 are in Standby state and therefore you must transition one of the NameNode to Active state. This transition can be performed using one of the following options:

- **Option I - Using CLI**

  Use the command line interface (CLI) to transition one of the NameNode to Active State. Execute the following command on that NameNode host machine:

  ```
  hdfs haadmin -failover --forcefence --forceactive <serviceId> <namenodeId>
  ```

  For more information on the `haadmin` command, see Appendix - Administrative Commands section in this document.

- **Option II - Deploying Automatic Failover**

  You can configure and deploy automatic failover using the instructions provided here.

1.4. Operating an HA cluster

- While operating an HA cluster, the Active NameNode cannot commit a transaction if it cannot write successfully to a quorum of the JournalNodes.
- When restarting an HA cluster, the steps for initializing JournalNodes and NN2 can be skipped.
- Start the services in the following order:

  1. JournalNodes
  2. NameNodes

  Note

  Verify that the ZKFailoverController (ZKFC) process on each node is running so that one of the NameNodes can be converted to active state.

  3. DataNodes

1.5. Configure and Deploy Automatic Failover

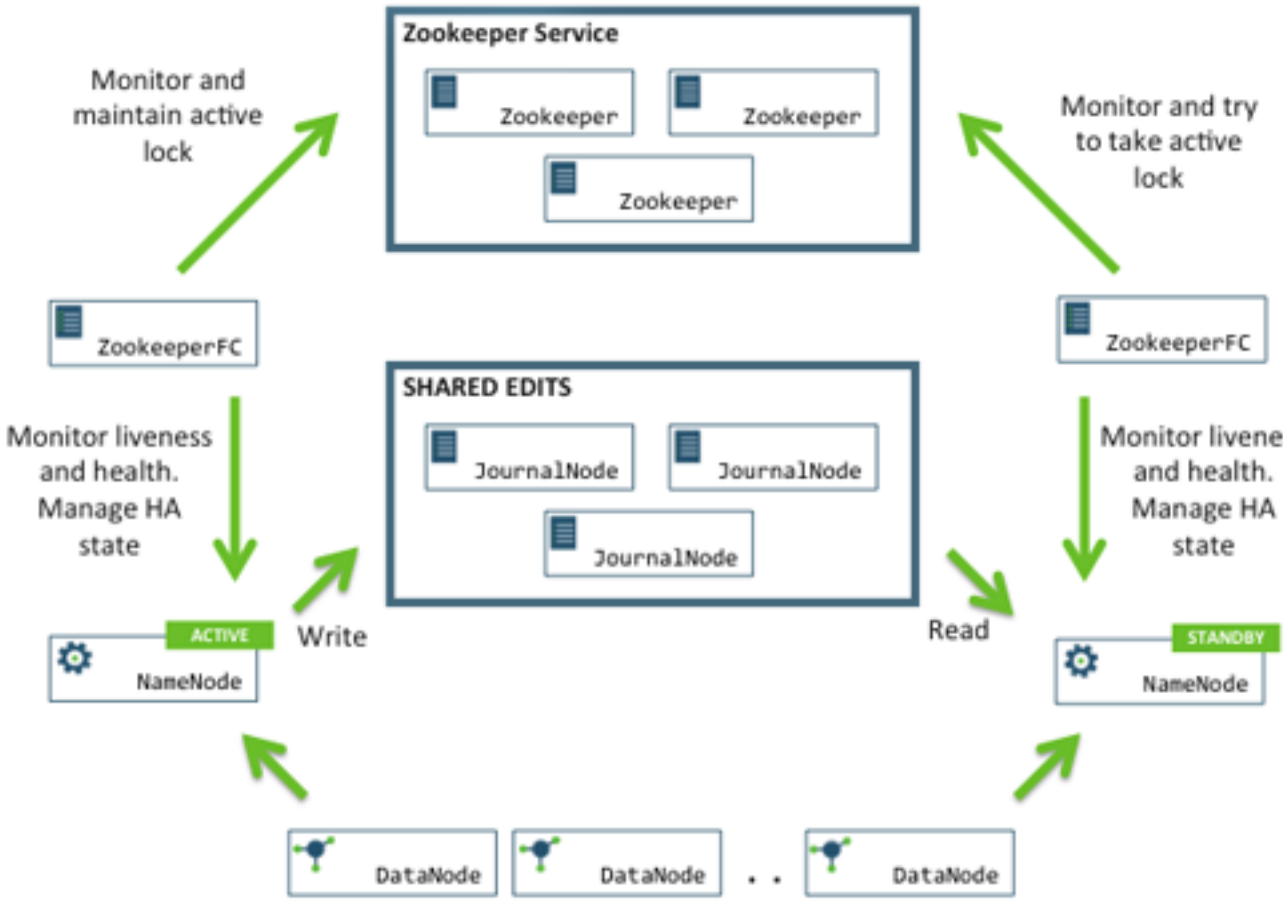
The above sections describe how to configure manual failover. In that mode, the system will not automatically trigger a failover from the active to the standby NameNode, even if the active node has failed. This section describes how to configure and deploy automatic failover.

Automatic failover adds following components to an HDFS deployment:

- ZooKeeper quorum
- ZKFailoverController process (abbreviated as ZKFC).

The ZKFailoverController (ZKFC) is a ZooKeeper client that monitors and manages the state of the NameNode. Each of the machines which run NameNode service also runs a ZKFC. ZKFC is responsible for:

- **Health monitoring:** ZKFC periodically pings its local NameNode with a health-check command.
- **ZooKeeper session management:** When the local NameNode is healthy, the ZKFC holds a session open in ZooKeeper. If the local NameNode is active, it also holds a special "lock" znode. This lock uses ZooKeeper's support for "ephemeral" nodes; if the session expires, the lock node will be automatically deleted.
- **ZooKeeper-based election:** If the local NameNode is healthy and no other node currently holds the lock znode, ZKFC will try to acquire the lock. If ZKFC succeeds, then it has "won the election" and will be responsible for running a failover to make its local NameNode active. The failover process is similar to the manual failover described above: first, the previous active is fenced if necessary and then the local NameNode transitions to active state.



Use the following instructions to configure and deploy automatic failover:

- Prerequisites
- Instructions

1.5.1. Prerequisites

Complete the following prerequisites:

- Ensure you have a working Zookeeper service. If you had an Ambari deployed HDP cluser with Zookeeper, you can use that. If not, deploy ZooKeeper using the instructions provided here.

Note

In a typical deployment, ZooKeeper daemons are configured to run on three or five nodes. It is however acceptable to collocate the ZooKeeper nodes on the same hardware as the HDFS NameNode and Standby Node. Many operators choose to deploy the third ZooKeeper process on the same node as the YARN ResourceManager. To achieve performance and improve isolation, Hortonworks recommends configuring the ZooKeeper nodes such that the ZooKeeper data and HDFS metadata is stored on separate disk drives.

- Shut down your HA cluster (configured for manual failover) using the instructions provided here.

  Ability to transition from a manual failover setup to an automatic failover setup, while the cluster is running, is not supported currently.

1.5.2. Instructions

Complete the following instructions:

1. Configure automatic failover.

   a. Set up your cluster for automatic failover.

   Add the following property to the the `hdfs-site.xml` file for both the NameNode machines:

```
<property>
   <name>dfs.ha.automatic-failover.enabled</name>
   <value>true</value>
 </property>
```

   b. List the host-port pairs running the ZooKeeper service.

   Add the following property to the the `core-site.xml` file for both the NameNode machines:

```
<property>
   <name>ha.zookeeper.quorum</name>
   <value>zk1.example.com:2181,zk2.example.com:2181,zk3.example.
com:2181</value>
 </property>
```

Note

Suffix the configuration key with the nameservice ID to configure the above settings on a per-nameservice basis. For example, in a cluster with federation enabled, you can explicitly enable automatic failover for only one of the nameservices by setting `dfs.ha.automatic-failover.enabled.`*`$my-nameservice-id`*.

2. Initialize HA state in ZooKeeper.

   Execute the following command on the NameNode hosts:

   ```
   hdfs zkfc -formatZK
   ```

   This command creates a `znode` in ZooKeeper. The automatic failover system stores uses this znode for data storage.

3. Check to see if Zookeeper is running. If not, start Zookeeper by executing the following command on the ZooKeeper host machine(s).

   ```
   su - zookeeper -c "export  ZOOCFGDIR=/etc/zookeeper/conf ; export ZOOCFG=
   zoo.cfg ; source /etc/zookeeper/conf/zookeeper-env.sh ; /usr/lib/zookeeper/
   bin/zkServer.sh start"
   ```

4. Start the JournalNodes, NameNodes, and DataNodes using the instructions provided here.

5. Start ZKFC.

   Manually start the `zkfc` daemon on each of the NameNode host machines using the following command:

   ```
   /usr/lib/hadoop/sbin/hadoop-daemon.sh start zkfc
   ```

   The sequence of starting ZKFC determines which NameNode will become Active. For example, if ZKFC is started on NN1 first, it will cause NN1 to become Active.

Note

To convert a non-HA cluster to an HA cluster, Hortonworks recommends that you run the `bootstrapStandby` command (this command is used to initialize NN2) **before** you start ZKFC on any of the NameNode machines.

6. Verify automatic failover.

   a. Locate the Active NameNode.

      Use the NameNode web UI to check the status for each NameNode host machine.

   b. Cause a failure on the Active NameNode host machine.

      For example, you can use the following command to simulate a JVM crash:

      ```
      kill -9 $PID_of_Active_NameNode
      ```

Or, you could power cycle the machine or unplug its network interface to simulate outage.

c. The Standby NameNode should now automatically become Active within several seconds.

Note

The amount of time required to detect a failure and trigger a failover depends on the configuration of `ha.zookeeper.session-timeout.ms` property (default value is 5 seconds).

d. If the test fails, your HA settings might be incorrectly configured.

Check the logs for the zkfc daemons and the NameNode daemons to diagnose the issue.

1.6. Appendix - Administrative Commands

The subcommands of `hdfs haadmin` are extensively used for administering an HA cluster.

Running the `hdfs haadmin` command without any additional arguments will display the following usage information:

```
Usage: DFSHAAdmin [-ns <nameserviceId>]
    [-transitionToActive <serviceId>]
    [-transitionToStandby <serviceId>]
    [-failover [--forcefence] [--forceactive] <serviceId> <serviceId>]
    [-getServiceState <serviceId>]
    [-checkHealth <serviceId>]
    [-help <command>
```

This section provides high-level uses of each of these subcommands.

- **transitionToActive** and **transitionToStandby**: Transition the state of the given NameNode to Active or Standby.

  These subcommands cause a given NameNode to transition to the Active or Standby state, respectively. These commands do not attempt to perform any fencing, and thus should be used **rarely**. Instead, Hortonworks recommends using the following subcommand:

  ```
  hdfs haadmin -failover
  ```

- **failover**: Initiate a failover between two NameNodes.

  This subcommand causes a failover from the first provided NameNode to the second.

  - If the first NameNode is in the Standby state, this command transitions the second to the Active state without error.
  - If the first NameNode is in the Active state, an attempt will be made to gracefully transition it to the Standby state. If this fails, the fencing methods (as configured by **dfs.ha.fencing.methods**) will be attempted in order until one succeeds. Only after this

process will the second NameNode be transitioned to the Active state. If the fencing methods fail, the second NameNode is not transitioned to Active state and an error is returned.

- **getServiceState**: Determine whether the given NameNode is Active or Standby.

  This subcommand connects to the provided NameNode, determines its current state, and prints either "standby" or "active" to STDOUT appropriately. This subcommand might be used by cron jobs or monitoring scripts.

- **checkHealth**: Check the health of the given NameNode.

  This subcommand connects to the NameNode to check its health. The NameNode is capable of performing some diagnostics that include checking if internal services are running as expected. This command will return 0 if the NameNode is healthy else it will return a non-zero code.

  Note

  This subcommand is in implementation phase and currently always returns success unless the given NameNode is down.

2. High Availability for Hive Metastore

This document is intended for system administrators who need to configure the Hive Metastore service for High Availability.

To learn more, see HDP's Full-Stack HA Architecture.

2.1. Use Cases and Fail Over Scenarios

This section provides information on the use cases and fail over scenarios for high availability (HA) in the Hive metastore.

Use Cases

The metastore HA solution is designed to handle metastore service failures. Whenever a deployed metastore service goes down, metastore service can remain unavailable for a considerable time until service is brought back up. To avoid such outages, deploy the metastore service in HA mode.

Deployment Scenarios

We recommend deploying the metastore service on multiple boxes concurrently. Each Hive metastore client will read the configuration property `hive.metastore.uris` to get a list of metastore servers with which it can try to communicate.

```
<property>
 <name> hive.metastore.uris </name>
 <value> thrift://$Hive_Metastore_Server_Host_Machine_FQDN </value>
 <description> A comma separated list of metastore uris on which metastore
 service is running </description>
 </property>
```

These metastore servers store their state in a MySQL HA cluster, which should be set up as recommended in the whitepaper *"MySQL Replication for Failover Protection."*

In the case of a secure cluster, each of the metastore servers will additionally need to have the following configuration property in its `hive-site.xml` file.

```
<property>
 <name> hive.cluster.delegation.token.store.class </name>
 <value> org.apache.hadoop.hive.thrift.DBTokenStore </value>
 </property>
```

Fail Over Scenario

A Hive metastore client always uses the first URI to connect with the metastore server. In case the metastore server becomes unreachable, the client will randomly pick up a URI from the list and try connecting with that.

2.2. Software Configuration

Complete the following tasks to configure Hive HA solution:

- Install HDP
- Update the Hive Metastore
- Validate configuration

2.2.1. Install HDP

Use the following instructions to install HDP on your cluster hardware. Ensure that you specify the virtual machine (configured in the previous section) as your NameNode.

1. Download Apache Ambari using the instructions provided here.

Note

Do not start the Ambari server until you have configured the relevant templates as outlined in the following steps.

2. Edit the `<master-install-machine-for-Hive-Metastore>/etc/hive/conf.server/hive-site.xml` configuration file to add the following properties:

   a. Provide the URI for the client to contact Metastore server. The following property can have a comma separated list when your cluster has multiple Hive Metastore servers.

```
<property>
 <name> hive.metastore.uris </name>
 <value> thrift://$Hive_Metastore_Server_Host_Machine_FQDN </value>
 <description> URI for client to contact metastore server </description>
</property>
```

   b. Configure Hive cluster delegation token storage class.

```
<property>
 <name> hive.cluster.delegation.token.store.class </name>
 <value> org.apache.hadoop.hive.thrift.DBTokenStore </value>
 </property>
```

3. Complete HDP installation.

   - Continue the Ambari installation process using the instructions provided here.
   - Complete the Ambari installation. Ensure that the installation was successful.

2.2.2. Update the Hive Metastore

HDP components configured for HA must use a NameService rather than a NameNode. Use the following instructions to update the Hive Metastore to reference the NameService rather than a Name Node.

Note

Hadoop administrators also often use the following procedure to update the Hive metastore with the new URI for a node in a Hadoop cluster. For example, administrators sometimes rename an existing node as their cluster grows.

1. Open a command prompt on the machine hosting the Hive metastore.
2. Execute the following command to retrieve a list of URIs for the filesystem roots, including the location of the NameService:

```
hive --service metatool -listFSRoot
```

3. Execute the following command with the `-dryRun` option to test your configuration change before implementing it:

```
hive --service metatool -updateLocation <nameservice-uri> <namenode-uri> -
dryRun
```

4. Execute the command again, this time without the `-dryRun` option:

```
hive --service metatool -updateLocation <nameservice-uri> <namenode-uri>
```

2.2.3. Validate configuration

Test various fail over scenarios to validate your configuration.

3. WebHDFS Administrator Guide

Use the following instructions to set upo WebHDFS:

1. Set up WebHDFS.

   Add the following property to the `hdfs-site.xml` file:

```
<property>
   <name>dfs.webhdfs.enabled</name>
   <value>true</value>
</property>
```

2. [Optional] - If running a secure cluster, follow the steps listed below.

   a. Create an HTTP service user principal using the command given below:

```
kadmin: addprinc -randkey HTTP/$<Fully_Qualified_Domain_Name>@
$<Realm_Name>.COM
```

   where:

   - `Fully_Qualified_Domain_Name`: Host where NameNode is deployed
   - `Realm_Name`: Name of your Kerberos realm

   b. Create keytab files for the HTTP principals.

```
kadmin: xst -norandkey -k /etc/security/spnego.service.keytab HTTP/
$<Fully_Qualified_Domain_Name>
```

   c. Verify that the keytab file and the principal are associated with the correct service.

```
klist –k -t /etc/security/spnego.service.keytab
```

   d. Add the following properties to the `hdfs-site.xml` file.

```
<property>
   <name>dfs.web.authentication.kerberos.principal</name>
   <value>HTTP/$<Fully_Qualified_Domain_Name>@$<Realm_Name>.COM</value>
</property>
```

```
<property>
   <name>dfs.web.authentication.kerberos.keytab</name>
   <value>/etc/security/spnego.service.keytab</value>
</property>
```

   where:

   - `Fully_Qualified_Domain_Name`: Host where NameNode is deployed
   - `Realm_Name`: Name of your Kerberos realm

3. Restart the NameNode and DataNode services using the instructions provided here.