# Table of Contents

## About this Guide ........................................................................................................5

## Configuring HDFS High Availability..........................................................................7

- Introduction to HDFS High Availability........................................................................7
  - Overview.......................................................................................................................7
  - Background..................................................................................................................7
  - Architecture...............................................................................................................7
- Configuring Hardware for HDFS HA................................................................................8
  - Hardware Configuration for Quorum-based Storage......................................................8
- Configuring Software for HDFS HA................................................................................9
- Deploying HDFS High Availability................................................................................17
  - Install and Start the JournalNodes................................................................................17
  - Format the NameNode (if new cluster)........................................................................18
  - Initialize the Shared Edits directory (if converting existing non-HA cluster)..............18
  - Start the NameNodes....................................................................................................18
  - Restart Services (if converting existing non-HA cluster)............................................18
  - Deploy Automatic Failover (if it is configured)............................................................19
  - Verifying Automatic Failover......................................................................................20
- Upgrading an HDFS HA Configuration to the Latest Release......................................20
  - Upgrading to CDH 5 ....................................................................................................20
- Configuring Other CDH Components to Use HDFS HA................................................20
  - Configuring HBase to Use HDFS HA.........................................................................20
  - Configuring Oozie to Use HDFS HA.........................................................................21
  - Upgrading the Hive Metastore to Use HDFS HA.........................................................21
- Administering an HDFS High Availability Cluster........................................................22
  - Using the haadmin command......................................................................................22
  - Using the dfsadmin command when HA is enabled.....................................................23
  - Moving an HA NameNode to a New Host...................................................................23
  - Disabling HDFS High Availability.............................................................................25
  - Redeploying HDFS High Availability.......................................................................29

## Configuring MapReduce High Availability (MRv1 or MRv2).................................31

- Configuring High Availability for ResourceManager (MRv2/YARN)............................31
  - Architecture...............................................................................................................31
  - Setting up ResourceManager HA.................................................................................32
  - Using `yarn rmadmin` to Administer ResourceManager HA......................................37
Configuring High Availability for Other CDH Components

About Oozie High Availability
Configuring High Availability for Llama
Fencing
Reclaiming Cluster Resources
Configuring HA
About this Guide

This *CDH 5 High Availability Guide* is for Apache Hadoop system administrators who want to enable continuous availability by configuring a CDH 5 cluster without single points of failure.

The guide explains the following topics:

- [Configuring HDFS High Availability](#) on page 7
- [Configuring High Availability for the JobTracker (MRv1)](#) on page 37
- [Configuring High Availability for ResourceManager (MRv2/YARN)](#) on page 31
- [About Oozie High Availability](#) on page 51
Configuring HDFS High Availability

This section explains how to configure a highly-available NameNode. The following topics provide more information and instructions:

- Introduction to HDFS High Availability
- Configuring Hardware for HDFS HA on page 8
- Configuring Software for HDFS HA on page 9
- Deploying HDFS High Availability on page 17
- Upgrading an HDFS HA Configuration to the Latest Release
- Configuring other Components to use HDFS HA
- Administering an HDFS High Availability Cluster on page 22

Introduction to HDFS High Availability

- Overview
- Background
- Architecture

Overview

This guide provides an overview of the HDFS High Availability (HA) feature and how to configure and manage an HA HDFS cluster.

This document assumes that the reader has a general understanding of components and node types in an HDFS cluster. For details, see the Apache HDFS Architecture Guide.

Background

In earlier releases, 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, the cluster as a whole would be unavailable until the NameNode was either restarted or brought up on a separate machine. The Secondary NameNode did not provide failover capability.

This reduced the total availability of the HDFS cluster in two major ways:

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

The HDFS HA feature addresses the above problems by providing the option of running two NameNodes in the same cluster, in an Active/Passive configuration. These are referred to as the Active NameNode and the Standby NameNode. Unlike the Secondary NameNode, the Standby NameNode is hot standby, allowing a fast failover to a new NameNode in the case that a machine crashes, or a graceful administrator-initiated failover for the purpose of planned maintenance. You cannot have more than two NameNodes.

Architecture

In a typical HA cluster, two separate machines are configured as NameNodes. At any point in time, one of the NameNodes is in an Active state, and the other is in a Standby state. The Active NameNode is responsible for all client operations in the cluster, while the Standby is simply acting as a slave, maintaining enough state to provide a fast failover if necessary.
Quorum-based Storage

Quorum-based Storage refers to the HA implementation that uses Quorum Journal Manager (QJM).

- **Note:**
  Quorum-based Storage is the only implementation Cloudera supports in CDH 5.

In order for the Standby node to keep its state synchronized with the Active node in this implementation, both nodes communicate with a group of separate daemons called JournalNodes. When any namespace modification is performed by the Active node, it durably logs a record of the modification to a majority of these JournalNodes. The Standby node is capable of reading the edits from the JournalNodes, and is constantly watching them for changes to the edit log. As the Standby Node sees the edits, it applies them to its own namespace. In the event of a failover, the Standby will ensure that it has read all of the edits from the JournalNodes before promoting itself to the Active state. This ensures that the namespace state is fully synchronized before a failover occurs.

In order to provide a fast failover, it is also necessary that the Standby node has up-to-date information regarding the location of blocks in the cluster. In order to achieve this, the DataNodes are configured with the location of both NameNodes, and they send block location information and heartbeats to both.

It is vital for the correct operation of an HA cluster that only one of the NameNodes be active at a time. Otherwise, the namespace state would quickly diverge between the two, risking data loss or other incorrect results. In order to ensure this property and prevent the so-called "split-brain scenario," the JournalNodes will only ever allow a single NameNode to be a writer at a time. During a failover, the NameNode which is to become active will simply take over the role of writing to the JournalNodes, which will effectively prevent the other NameNode from continuing in the Active state, allowing the new Active NameNode to safely proceed with failover.

- **Note:** Because of this, fencing is not required, but it is still useful; see Configuring Software for HDFS HA on page 9.

Configuring Hardware for HDFS HA

- **Note:**
  Quorum-based Storage is the only HDFS HA implementation supported in CDH 5.

Hardware Configuration for Quorum-based Storage

In order to deploy an HA cluster using Quorum-based Storage, you should prepare the following:

- **NameNode** machines - the machines on which you run the Active and Standby NameNodes should have equivalent hardware to each other, and equivalent hardware to what would be used in a non-HA cluster.
- **JournalNode** machines - the machines on which you run the JournalNodes.
- The JournalNode daemon is relatively lightweight, so these daemons can reasonably be collocated on machines with other Hadoop daemons, for example NameNodes, the JobTracker, or the YARN ResourceManager.
- Cloudera recommends that you deploy the JournalNode daemons on the "master" host or hosts (NameNode, Standby NameNode, JobTracker, etc.) so the JournalNodes' local directories can use the reliable local storage on those machines. You should not use SAN or NAS storage for these directories.
- There must be at least three JournalNode daemons, since edit log modifications must be written to a majority of JournalNodes. This will allow the system to tolerate the failure of a single machine. You can also run more than three JournalNodes, but in order to actually increase the number of failures the system can tolerate, you should run an odd number of JournalNodes, (three, five, seven, etc.) Note that when running with N JournalNodes, the system can tolerate at most (N - 1) / 2 failures and continue to function normally. If the
requisite quorum is not available, the NameNode will not format or start, and you will see an error similar to this:

```
12/10/01 17:34:18 WARN namenode.FSEditLog: Unable to determine input streams from QJM to [10.0.1.10:8485, 10.0.1.10:8486, 10.0.1.10:8487]. Skipping.
java.io.IOException: Timed out waiting 20000ms for a quorum of nodes to respond.
```

**Note:**

In an HA cluster, the Standby NameNode also performs checkpoints of the namespace state, and thus it is not necessary to run a Secondary NameNode, CheckpointNode, or BackupNode in an HA cluster. In fact, to do so would be an error. If you are reconfiguring a non-HA-enabled HDFS cluster to be HA-enabled, you can reuse the hardware which you had previously dedicated to the Secondary NameNode.

---

**Configuring Software for HDFS HA**

This section describes the software configuration required for HDFS HA.

**Note:**

The subsections that follow explain how to configure HDFS HA using Quorum-based storage. This is the only implementation supported in CDH 5.

---

**Configuration Overview**

As with HDFS Federation configuration, HA configuration is backward compatible and allows existing single NameNode configurations to work without change. The new configuration is designed such that all the nodes in the cluster can have the same configuration without the need for deploying different configuration files to different machines based on the type of the node.

HA clusters reuse the **NameService ID** to identify a single HDFS instance that may consist of multiple HA NameNodes. In addition, there is a new abstraction called **NameNode ID**. Each distinct NameNode in the cluster has a different NameNode ID. To support a single configuration file for all of the NameNodes, the relevant configuration parameters include the NameService ID as well as the NameNode ID.

**Changes to Existing Configuration Parameters**

The following configuration parameter has changed for YARN implementations:

`fs.defaultFS` - formerly `fs.default.name`, the default path prefix used by the Hadoop FS client when none is given. (`fs.default.name` is deprecated for YARN implementations, but will still work.)

 Optionally, you can configure the default path for Hadoop clients to use the HA-enabled logical URI. For example, if you use `mycluster` as the NameService ID as shown below, this will be the value of the authority portion of all of your HDFS paths. You can configure the default path in your `core-site.xml` file:

```
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://mycluster</value>
</property>
```
Configuring HDFS High Availability

- For MRv1:

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

New Configuration Parameters

To configure HA NameNodes, you must add several configuration options to your `hdfs-site.xml` configuration file.

The order in which you set these configurations is unimportant, but the values you choose for `dfs.nameservices` and `dfs.ha.namenodes.[NameService ID]` will determine the keys of those that follow. This means that you should decide on these values before setting the rest of the configuration options.

Configure `dfs.nameservices`

dfs.nameservices - the logical name for this new nameservice

Choose a logical name for this nameservice, for example `mycluster`, and use this logical name for the value of this configuration option. The name you choose is arbitrary. It will be used both for configuration and as the authority component of absolute HDFS paths in the cluster.

- **Note:**
  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.

```
<property>
  <name>dfs.nameservices</name>
  <value>mycluster</value>
</property>
```

Configure `dfs.ha.namenodes.[nameservice ID]`

dfs.ha.namenodes.[nameservice ID] - unique identifiers for each NameNode in the nameservice

Configure a list of comma-separated NameNode IDs. This will be used by DataNodes to determine all the NameNodes in the cluster. For example, if you used `mycluster` as the NameService ID previously, and you wanted to use `nn1` and `nn2` as the individual IDs of the NameNodes, you would configure this as follows:

```
<property>
  <name>dfs.ha.namenodes.mycluster</name>
  <value>nn1,nn2</value>
</property>
```

- **Note:**
  In this release, you can configure a maximum of two NameNodes per nameservice.

Configure `dfs.namenode.rpc-address.[nameservice ID]`

dfs.namenode.rpc-address.[nameservice ID].[name node ID] - the fully-qualified RPC address for each NameNode to listen on

For both of the previously-configured NameNode IDs, set the full address and RPC port of the NameNode process. Note that this results in two separate configuration options. For example:

```
<property>
  <name>dfs.namenode.rpc-address.mycluster.nn1</name>
</property>
```
Configure dfs.namenode.http-address.[nameservice ID]

dfs.namenode.http-address.[nameservice ID].[name node ID] - the fully-qualified HTTP address for each NameNode to listen on

Similarly to rpc-address above, set the addresses for both NameNodes' HTTP servers to listen on. For example:

```xml
<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's Kerberos security features enabled, and you intend to use HSFTP, you should also set the https-address similarly for each NameNode.

Configure dfs.namenode.shared.edits.dir

dfs.namenode.shared.edits.dir - the location of the shared storage directory

Configure the addresses of the JournalNodes which provide the shared edits storage, written to by the Active NameNode and read by the Standby NameNode to stay up-to-date with all the file system changes the Active NameNode makes. Though you must specify several JournalNode addresses, you should only configure one of these URIs. The URI should be in 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. Though it is not a requirement, it's a good idea to reuse the nameservice ID for the journal identifier.

For example, if the JournalNodes for this cluster were running on the machines node1.example.com, node2.example.com, and node3.example.com, and the nameservice ID were mycluster, you would use the following as the value for this setting (the default port for the JournalNode is 8485):

```xml
<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>
```

Configure dfs.journalnode.edits.dir

dfs.journalnode.edits.dir - the path where the JournalNode daemon will store its local state
On each JournalNode machine, configure the absolute path where the edits and other local state information used by the JournalNodes will be stored; use only a single path per JournalNode. (The other JournalNodes provide redundancy; you can also configure this directory on a locally-attached RAID-1 or RAID-10 array.)

For example:

```
<property>
  <name>dfs.journalnode.edits.dir</name>
  <value>/data/1/dfs/jn</value>
</property>
```

Now create the directory (if it doesn't already exist) and make sure its owner is hdfs, for example:

```
$ sudo mkdir -p /data/1/dfs/jn
$ sudo chown -R hdfs:hdfs /data/1/dfs/jn
```

**Client Failover Configuration**

dfs.client.failover.proxy.provider.[nameservice ID] - the Java class that HDFS clients use to contact the Active NameNode

Configure the name of the Java class which the DFS Client will use to determine which NameNode is the current Active, and therefore which NameNode is currently serving client requests. The only implementation which currently ships with Hadoop is the `ConfiguredFailoverProxyProvider`, so use this unless you are using a custom one. For example:

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

**Fencing Configuration**

dfs.ha.fencing.methods - a list of scripts or Java classes which will be used to fence the Active NameNode during a failover

It is desirable for correctness of the system that only one NameNode be in the Active state at any given time.

- **Important:**
  
  When you use Quorum-based Storage, only one NameNode will ever be allowed to write to the JournalNodes, so there is no potential for corrupting the file system metadata in a “split-brain” scenario. But when a failover occurs, it is still possible that the previously active NameNode could serve read requests to clients - and these requests may be out of date - until that NameNode shuts down when it tries to write to the JournalNodes. For this reason, it is still desirable to configure some fencing methods even when using Quorum-based Storage.

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:**

  If you choose to use no actual fencing methods, you still must configure something for this setting, for example `shell(/bin/true)`.

The fencing methods used during a failover are configured as a carriage-return-separated list, and these will be attempted in order until one of them indicates that fencing has succeeded.

There are two fencing methods which ship with Hadoop:
- **sshfence**
- **shell**

For information on implementing your own custom fencing method, see the `org.apache.hadoop.ha.NodeFencer` class.

### Configuring the sshfence fencing method

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

The sshfence option uses SSH to connect 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. Thus, you must also configure the `dfs.ha.fencing.ssh.private-key-files` option, which is a comma-separated list of SSH private key files.

**Important:**

The files must be accessible to the user running the NameNode processes (typically the hdfs user on the NameNode hosts).

For example:

```xml
<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 configure a non-standard username or port to perform the SSH as shown below. You can also configure a timeout, in milliseconds, for the SSH, after which this fencing method will be considered to have failed:

```xml
<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>
  <description>
    SSH connection timeout, in milliseconds, to use with the builtin sshfence fencer.
  </description>
</property>
```

### Configuring the shell fencing method

**shell** - run an arbitrary shell command to fence the Active NameNode

The shell fencing method runs an arbitrary shell command, which you can configure as shown below:

```xml
<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 cannot include any closing parentheses.

When executed, the first argument to the configured script will be the address of the NameNode to be fenced, followed by all arguments specified in the configuration.
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`.

The following variables referring to the target node to be fenced are also available:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$target_host</td>
<td>Hostname of the node to be fenced</td>
</tr>
<tr>
<td>$target_port</td>
<td>IPC port of the node to be fenced</td>
</tr>
<tr>
<td>$target_address</td>
<td>The above two variables, combined as <a href="">host:port</a></td>
</tr>
<tr>
<td>$target_nameserviceid</td>
<td>The nameservice ID of the NameNode to be fenced</td>
</tr>
<tr>
<td>$target_namenodeid</td>
<td>The namenode ID of the NameNode to be fenced</td>
</tr>
</tbody>
</table>

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

```xml
<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 determined to be successful. If it returns any other exit code, the fencing was not successful and the next fencing method in the list will be attempted.

**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).

### Automatic Failover Configuration

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.

### Component Overview

Automatic failover adds two new components to an HDFS deployment: a ZooKeeper quorum, and the ZKFailoverController process (abbreviated as ZKFC).

Apache ZooKeeper is a highly available service for maintaining small amounts of coordination data, notifying clients of changes in that data, and monitoring clients for failures. The implementation of automatic HDFS failover relies on ZooKeeper for the following things:

- **Failure detection** - each of the NameNode machines in the cluster maintains a persistent session in ZooKeeper. If the machine crashes, the ZooKeeper session will expire, notifying the other NameNode that a failover should be triggered.
- **Active NameNode election** - ZooKeeper provides a simple mechanism to exclusively elect a node as active. If the current active NameNode crashes, another node can take a special exclusive lock in ZooKeeper indicating that it should become the next active NameNode.
The **ZKFailoverController** (ZKFC) is a new component - a ZooKeeper client which also monitors and manages the state of the NameNode. Each of the machines which runs a NameNode also runs a ZKFC, and that ZKFC is responsible for:

- **Health monitoring** - the ZKFC pings its local NameNode on a periodic basis with a health-check command. So long as the NameNode responds promptly with a healthy status, the ZKFC considers the node healthy. If the node has crashed, frozen, or otherwise entered an unhealthy state, the health monitor will mark it as unhealthy.

- **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 the ZKFC sees that no other node currently holds the lock `znode`, it will itself try to acquire the lock. If it succeeds, then it has "won the election", and is 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.

**Deploying ZooKeeper**

In a typical deployment, ZooKeeper daemons are configured to run on three or five nodes. Since ZooKeeper itself has light resource requirements, it is acceptable to collocate the ZooKeeper nodes on the same hardware as the HDFS NameNode and Standby Node. Operators using MapReduce v2 (MRv2) often choose to deploy the third ZooKeeper process on the same node as the YARN ResourceManager. It is advisable to configure the ZooKeeper nodes to store their data on separate disk drives from the HDFS metadata for best performance and isolation.

See the [ZooKeeper documentation](#) for instructions on how to set up a ZooKeeper ensemble. In the following sections we assume that you have set up a ZooKeeper cluster running on three or more nodes, and have verified its correct operation by connecting using the ZooKeeper command-line interface (CLI).

**Configuring Automatic Failover**

**Note:**

Before you begin configuring automatic failover, you must shut down your cluster. It is not currently possible to transition from a manual failover setup to an automatic failover setup while the cluster is running.

Configuring automatic failover requires two additional configuration parameters. In your `hdfs-site.xml` file, add:

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

This specifies that the cluster should be set up for automatic failover. In your `core-site.xml` file, add:

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

This lists the host-port pairs running the ZooKeeper service.

As with the parameters described earlier in this document, these settings may be configured on a per-nameservice basis by suffixing the configuration key with the nameservice ID. 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`.
There are several other configuration parameters which you can set to control the behavior of automatic failover, but they are not necessary for most installations. See the configuration section of the Hadoop documentation for details.

**Initializing the HA state in ZooKeeper**

After you have added the configuration keys, the next step is to initialize the required state in ZooKeeper. You can do so by running the following command from one of the NameNode hosts.

```
$hdfs zkfc -formatZK
```

This will create a znode in ZooKeeper in which the automatic failover system stores its data.

**Securing access to ZooKeeper**

If you are running a secure cluster, you will probably want to ensure that the information stored in ZooKeeper is also secured. This prevents malicious clients from modifying the metadata in ZooKeeper or potentially triggering a false failover.

In order to secure the information in ZooKeeper, first add the following to your `core-site.xml` file:

```xml
<property>
  <name>ha.zookeeper.auth</name>
  <value>@/path/to/zk-auth.txt</value>
</property>
<property>
  <name>ha.zookeeper.acl</name>
  <value>@/path/to/zk-acl.txt</value>
</property>
```

Note the '@' character in these values – this specifies that the configurations are not inline, but rather point to a file on disk.

The first configured file specifies a list of ZooKeeper authentications, in the same format as used by the ZooKeeper CLI. For example, you may specify something like `digest:hdfs-zkfcs:mypassword` where `hdfs-zkfcs` is a unique username for ZooKeeper, and `mypassword` is some unique string used as a password.

Next, generate a ZooKeeper Access Control List (ACL) that corresponds to this authentication, using a command such as the following:

```
$ java -cp $ZK_HOME/lib/*:$ZK_HOME/zookeeper-3.4.2.jar org.apache.zookeeper.server.auth.DigestAuthenticationProvider hdfs-zkfcs:mypassword
output: hdfs-zkfcs:mypassword->hdfs-zkfcs:P/OQvnYyU/7F/mGYvB/xurX8dYs=
```

Copy and paste the section of this output after the '->' string into the file `zk-acls.txt`, prefixed by the string "digest". For example:

```
digest:hdfs-zkfcs:v1UvLnd8MlacsE80rDuu6ONESbM=:rwda
```

To put these ACLs into effect, rerun the `zkfc -formatZK` command as described above.

After doing so, you can verify the ACLs from the ZooKeeper CLI as follows:

```
[zk: localhost:2181(CONNECTED) 1] getAcl /hadoop-ha
'digest','hdfs-zkfcs:v1UvLnd8MlacsE80rDuu6ONESbM=:cdrwa
```
Automatic Failover FAQ

Is it important that I start the ZKFC and NameNode daemons in any particular order?

No. On any given node you may start the ZKFC before or after its corresponding NameNode.

What additional monitoring should I put in place?

You should add monitoring on each host that runs a NameNode to ensure that the ZKFC remains running. In some types of ZooKeeper failures, for example, the ZKFC may unexpectedly exit, and should be restarted to ensure that the system is ready for automatic failover. Additionally, you should monitor each of the servers in the ZooKeeper quorum. If ZooKeeper crashes, automatic failover will not function.

What happens if ZooKeeper goes down?

If the ZooKeeper cluster crashes, no automatic failovers will be triggered. However, HDFS will continue to run without any impact. When ZooKeeper is restarted, HDFS will reconnect with no issues.

Can I designate one of my NameNodes as primary/preferred?

No. Currently, this is not supported. Whichever NameNode is started first will become active. You may choose to start the cluster in a specific order such that your preferred node starts first.

How can I initiate a manual failover when automatic failover is configured?

Even if automatic failover is configured, you can initiate a manual failover using the `hdfs haadmin -failover` command. It will perform a coordinated failover.

Deploying HDFS High Availability

After you have set all of the necessary configuration options, you are ready to start the JournalNodes and the two HA NameNodes.

Important: Before you start:

Make sure you have performed all the configuration and setup tasks described under Configuring Hardware for HDFS HA on page 8 and Configuring Software for HDFS HA on page 9, including initializing the HA state in ZooKeeper if you are deploying automatic failover.

Install and Start the JournalNodes

1. Install the JournalNode daemons on each of the machines where they will run.

   To install JournalNode on Red Hat-compatible systems:

   ```
   $ sudo yum install hadoop-hdfs-journalnode
   ```

   To install JournalNode on Ubuntu and Debian systems:

   ```
   $ sudo apt-get install hadoop-hdfs-journalnode
   ```

   To install JournalNode on SLES systems:

   ```
   $ sudo zypper install hadoop-hdfs-journalnode
   ```

2. Start the JournalNode daemons on each of the machines where they will run:

   ```
   sudo service hadoop-hdfs-journalnode start
   ```

   Wait for the daemons to start before formatting the primary NameNode (in a new cluster) and before starting the NameNodes (in all cases).
**Format the NameNode (if new cluster)**

If you are setting up a new HDFS cluster, format the NameNode you will use as your primary NameNode; see [Formatting the NameNode](#).

- **Important:** Make sure the JournalNodes have started

  Formatting will fail if, as instructed, you have configured the NameNode to communicate with the JournalNodes, but have not started the JournalNodes.

---

**Initialize the Shared Edits directory (if converting existing non-HA cluster)**

If you are converting a non-HA NameNode to HA, initialize the shared edits directory with the edits data from the local NameNode edits directories:

```bash
hdfs namenode -initializeSharedEdits
```

---

**Start the NameNodes**

1. Start the primary (formatted) NameNode:

   ```bash
   $ sudo service hadoop-hdfs-namenode start
   ```

2. Start the standby NameNode:

   ```bash
   $ sudo -u hdfs hdfs namenode -bootstrapStandby
   $ sudo service hadoop-hdfs-namenode start
   ```

- **Note:**

  If Kerberos is enabled, do not use commands in the form `sudo -u <user> <command>;` they will fail with a security error. Instead, use the following commands:

  ```bash
  $ kinit <user>
  ```

  (if you are using a password) or

  ```bash
  $ kinit -kt <keytab> <principal>
  ```

  (if you are using a keytab) and then, for each command executed by this user,

  ```bash
  $ <command>
  ```

Starting the standby NameNode with the `–bootstrapStandby` option copies over the contents of the primary NameNode's metadata directories (including the namespace information and most recent checkpoint) to the standby NameNode. (The location of the directories containing the NameNode metadata is configured via the configuration options `dfs.namenode.name.dir` and/or `dfs.namenode.edits.dir`.)

You can visit each NameNode's web page by browsing to its configured HTTP address. Notice that next to the configured address is the HA state of the NameNode (either "Standby" or "Active"). Whenever an HA NameNode starts and automatic failover is not enabled, it is initially in the Standby state. If automatic failover is enabled the first NameNode that is started will become active.

---

**Restart Services (if converting existing non-HA cluster)**

If you are converting from a non-HA to an HA configuration, you need to restart the JobTracker and TaskTracker (for MRv1, if used), or ResourceManager, NodeManager, and JobHistory Server (for YARN), and the DataNodes:

**On each DataNode:**

```bash
$ sudo service hadoop-hdfs-datanode start
```
On each TaskTracker system (MRv1):

```bash
$ sudo service hadoop-0.20-mapreduce-tasktracker start
```

On the JobTracker system (MRv1):

```bash
$ sudo service hadoop-0.20-mapreduce-jobtracker start
```

Verify that the JobTracker and TaskTracker started properly:

```bash
sudo jps | grep Tracker
```

On the ResourceManager system (YARN):

```bash
$ sudo service hadoop-yarn-resourcemanager start
```

On each NodeManager system (YARN; typically the same ones where DataNode service runs):

```bash
$ sudo service hadoop-yarn-nodemanager start
```

On the MapReduce JobHistory Server system (YARN):

```bash
$ sudo service hadoop-mapreduce-historyserver start
```

### Deploy Automatic Failover (if it is configured)

If you have configured automatic failover using the ZooKeeper FailoverController (ZKFC), you must install and start the zkfc daemon on each of the machines that runs a NameNode. Proceed as follows.

To install ZKFC on Red Hat-compatible systems:

```bash
$ sudo yum install hadoop-hdfs-zkfc
```

To install ZKFC on Ubuntu and Debian systems:

```bash
$ sudo apt-get install hadoop-hdfs-zkfc
```

To install ZKFC on SLES systems:

```bash
$ sudo zypper install hadoop-hdfs-zkfc
```

To start the zkfc daemon:

```bash
$ sudo service hadoop-hdfs-zkfc start
```

It is not important that you start the ZKFC and NameNode daemons in a particular order. On any given node you can start the ZKFC before or after its corresponding NameNode.

You should add monitoring on each host that runs a NameNode to ensure that the ZKFC remains running. In some types of ZooKeeper failures, for example, the ZKFC may unexpectedly exit, and should be restarted to ensure that the system is ready for automatic failover.

Additionally, you should monitor each of the servers in the ZooKeeper quorum. If ZooKeeper crashes, then automatic failover will not function. If the ZooKeeper cluster crashes, no automatic failovers will be triggered. However, HDFS will continue to run without any impact. When ZooKeeper is restarted, HDFS will reconnect with no issues.
Verifying Automatic Failover

After the initial deployment of a cluster with automatic failover enabled, you should test its operation. To do so, first locate the active NameNode. As mentioned above, you can tell which node is active by visiting the NameNode web interfaces.

Once you have located your active NameNode, you can cause a failure on that node. For example, you can use `kill -9 <pid of NN>` to simulate a JVM crash. Or you can power-cycle the machine or its network interface to simulate different kinds of outages. After you trigger the outage you want to test, the other NameNode should automatically become active within several seconds. The amount of time required to detect a failure and trigger a failover depends on the configuration of `ha.zookeeper.session-timeout.ms`, but defaults to 5 seconds.

If the test does not succeed, you may have a misconfiguration. Check the logs for the zkfc daemons as well as the NameNode daemons in order to further diagnose the issue.

Upgrading an HDFS HA Configuration to the Latest Release

Upgrading to CDH 5

**Important:**

NFS shared storage is not supported in CDH 5. If you are using an HDFS HA configuration using NFS shared storage, **disable the configuration** before you begin the upgrade. You can [redploy HDFS HA](#) using Quorum-based storage either before or after the upgrade.

To upgrade an HDFS HA configuration using Quorum-base storage from CDH 4 to the latest release, follow the directions for upgrading a cluster under [Upgrading to the Latest Version of CDH 5](#)

Configuring Other CDH Components to Use HDFS HA

You can use the HDFS High Availability NameNodes with other components of CDH, including HBase, Oozie, and Hive.

Configuring HBase to Use HDFS HA

To configure HBase to use HDFS HA, proceed as follows.

**Step 1: Shut Down the HBase Cluster**

To **shut HBase down gracefully, stop the Thrift server and clients, then stop the cluster**:

1. Stop the Thrift server and clients:
   ```
   sudo service hbase-thrift stop
   ```
2. Stop the cluster by shutting down the master and the region servers:
   - Use the following command on the master node:
     ```
     sudo service hbase-master stop
     ```
   - Use the following command on each node hosting a region server:
     ```
     sudo service hbase-regionserver stop
     ```
Step 2: Configure hbase.rootdir

Change the distributed file system URI in hbase-site.xml to the name specified in the dfs.nameservices property in hdfs-site.xml. The clients must also have access to hdfs-site.xml's dfs.client.* settings to properly use HA.

For example, suppose the HDFS HA property dfs.nameservices is set to ha-nn in hdfs-site.xml. To configure HBase to use the HA NameNodes, specify that same value as part of your hbase-site.xml's hbase.rootdir value:

```xml
<!-- Configure HBase to use the HA NameNode nameservice -->
<property>
  <name>hbase.rootdir</name>
  <value>hdfs://ha-nn/hbase</value>
</property>
```

Step 4: Restart HBase

1. Start the HBase Master
2. Start each of the HBase Region Servers

HBase-HDFS HA Troubleshooting

Problem: HMasters fail to start.

Solution: Check for this error in the hmaster logs:

```
java.lang.IllegalArgumentException: java.net.UnknownHostException: ha-nn
at org.apache.hadoop.security.SecurityUtil.buildTokenService(SecurityUtil.java:431)
at org.apache.hadoop.hdfs.NameNodeProxies.createNonHAProxy(NameNodeProxies.java:161)
at org.apache.hadoop.hdfs.NameNodeProxies.createProxy(NameNodeProxies.java:126)
...```

If so, verify that Hadoop's hdfs-site.xml and core-site.xml files are in your hbase/conf directory. This may be necessary if you put your configurations in non-standard places.

Configuring Oozie to Use HDFS HA

To configure an Oozie workflow to use HDFS HA, use the HA HDFS URI instead of the NameNode URI in the <name-node> element of the workflow.

Example:

```xml
<action name="mr-node">
  <map-reduce>
    <job-tracker>${jobTracker}</job-tracker>
    <name-node>hdfs://ha-nn</name-node>
    <value>hdfs://ha-nn/hbase</value>
</property>
```

where ha-nn is the value of dfs.nameservices in hdfs-site.xml.

Upgrading the Hive Metastore to Use HDFS HA

The Hive Metastore can be configured to use HDFS High Availability. See Hive Installation.

To configure the Hive metastore to use HDFS HA, change the records to reflect the location specified in the dfs.nameservices property, using the Hive metatool to obtain and change the locations.
Configuring HDFS High Availability

- **Note:** Before attempting to upgrade the Hive metastore to use HDFS HA, shut down the metastore and back it up to a persistent store.

If you are unsure which version of Avro SerDe is used, use both the `serdePropKey` and `tablePropKey` arguments. For example:

```bash
$ metatool -listFSRoot hdfs://oldnamenode.com/user/hive/warehouse
dx
$ metatool -updateLocation hdfs://nameservice1 hdfs://oldnamenode.com -tablePropKey avro.schema.url
-se serdePropKey schema.url
dx
$ metatool -listFSRoot hdfs://nameservice1/user/hive/warehouse
```

where:
- hdfs://oldnamenode.com/user/hive/warehouse identifies the NameNode location.
- hdfs://nameservice1 specifies the new location and should match the value of the `dfs.nameservices` property.
- `tablePropKey` is a table property key whose value field may reference the HDFS NameNode location and hence may require an update. To update the Avro SerDe schema URL, specify `avro.schema.url` for this argument.
- `serdePropKey` is a SerDe property key whose value field may reference the HDFS NameNode location and hence may require an update. To update the Haivvero schema URL, specify `schema.url` for this argument.

- **Note:** The Hive MetaTool is a best effort service that tries to update as many Hive metastore records as possible. If it encounters an error during the update of a record, it skips to the next record.

Administering an HDFS High Availability Cluster

Using the haadmin command

Now that your HA NameNodes are configured and started, you will have access to some additional commands to administer your HA HDFS cluster. Specifically, you should familiarize yourself with the subcommands of the `hdfs haadmin` command.

This page describes high-level uses of some important subcommands. For specific usage information of each subcommand, you should run `hdfs haadmin -help <command>`.

```
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 simply 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 of the methods succeeds. Only after this process will the second NameNode be transitioned to the Active state. If no fencing method succeeds, the second NameNode will not be transitioned to the Active state, and an error will be returned.

- **Note:**

  Running `hdfs haadmin -failover` from the command line works whether you have configured HA from the command line (as described in this document) or via Cloudera Manager. This means you can initiate a failover manually even if Cloudera Manager is unavailable.
**getServiceState**

*getServiceState* - determine whether the given NameNode is Active or Standby

Connect to the provided NameNode to determine its current state, printing either "standby" or "active" toSTDOUT as appropriate. This subcommand might be used by cron jobs or monitoring scripts which need to behave differently based on whether the NameNode is currently Active or Standby.

**checkHealth**

*checkHealth* - check the health of the given NameNode

Connect to the provided NameNode to check its health. The NameNode is capable of performing some diagnostics on itself, including checking if internal services are running as expected. This command will return 0 if the NameNode is healthy, non-zero otherwise. One might use this command for monitoring purposes.

- **Note:**
  The checkHealth command is not yet implemented, and at present will always return success, unless the given NameNode is completely down.

**Using the dfsadmin command when HA is enabled**

When you use the *dfsadmin* command with HA enabled, you should use the `-fs` option to specify a particular NameNode using the RPC address, or service RPC address, of the NameNode. Not all operations are permitted on a standby NameNode. If the specific NameNode is left unspecified, only the operations to set quotas (-setQuota, -clrQuota, -setSpaceQuota, -clrSpaceQuota), report basic file system information (-report), and check upgrade progress (-upgradeProgress) will failover and perform the requested operation on the active NameNode. The "refresh" options (-refreshNodes, -refreshServiceAcl, -refreshUserToGroupsMappings, and -refreshSuperUserGroupsConfiguration) must be run on both the active and standby NameNodes.

**Moving an HA NameNode to a New Host**

Use the following steps to move one of the NameNodes to a new host.

In this example, the current NameNodes are called *nn1* and *nn2*, and the new NameNode is *nn2-alt*. The example assumes that *nn2-alt* is already a member of this CDH 5 HA cluster, that automatic failover is configured and that a JournalNode on *nn2* is to be moved to *nn2-alt*, in addition to NameNode service itself.

The procedure moves the NameNode and JournalNode services from *nn2* to *nn2-alt*, reconfigures *nn1* to recognize the new location of the JournalNode, and restarts *nn1* and *nn2-alt* in the new HA configuration.

**Step 1: Make sure that nn1 is the active NameNode**

Make sure that the NameNode that is *not* going to be moved is active; in this example, *nn1* must be active. You can use the NameNodes' web UIs to see which is active; see Start the NameNodes on page 18.

If *nn1* is not the active NameNode, use the `hdfs haadmin -failover` command to initiate a failover from *nn2* to *nn1*:

```
hdfs haadmin -failover nn2 nnhdfs haadmin -failover nn2 nn1
```

**Step 2: Stop services on nn2**

Once you've made sure that the node to be moved is inactive, stop services on that node; in this example, stop services on *nn2*. Stop the NameNode, the ZKFC daemon if this an automatic-failover deployment, and the JournalNode if you are moving it. Proceed as follows.
Configuring HDFS High Availability

1. Stop the NameNode daemon:

   $ sudo service hadoop-hdfs-namenode stop

2. Stop the ZKFC daemon if it is running:

   $ sudo service hadoop-hdfs-zkfc stop

3. Stop the JournalNode daemon if it is running:

   $ sudo service hadoop-hdfs-journalnode stop

4. Make sure these services are not set to restart on boot. If you are not planning to use nn2 as a NameNode again, you may want remove the services.

Step 3: Install the NameNode daemon on nn2-alt

See the instructions for installing hadoop-hdfs-namenode in the CDH 5 Installation Guide under Step 3: Install CDH 5 with YARN or Step 4: Install CDH 5 with MRv1.

Step 4: Configure HA on nn2-alt

See Configuring Software for HDFS HA on page 9 for the properties to configure on nn2-alt in core-site.xml and hdfs-site.xml, and explanations and instructions. You should copy the values that are already set in the corresponding files on nn2.

- If you are relocating a JournalNode to nn2-alt, follow these directions to install it, but don't start it yet.
- If you are using automatic failover, make sure you follow the instructions for configuring the necessary properties on nn2-alt and initializing the HA state in Zookeeper.

   Note:
   You do not need to shut down the cluster to do this if automatic failover is already configured as your failover method; shutdown is required only if you are switching from manual to automatic failover.

Step 5: Copy the contents of the dfs.name.dir and dfs.journalnode.edits.dir directories to nn2-alt

Use rsync or a similar tool to copy the contents of the dfs.name.dir directory, and the dfs.journalnode.edits.dir directory if you are moving the JournalNode, from nn2 to nn2-alt.

Step 6: If you are moving a JournalNode, update dfs.namenode.shared.edits.dir on nn1

If you are relocating a JournalNode from nn2 to nn2-alt, update dfs.namenode.shared.edits.dir in hdfs-site.xml on nn1 to reflect the new hostname. See this section for more information about dfs.namenode.shared.edits.dir.

Step 7: If you are using automatic failover, install the zkfc daemon on nn2-alt

For instructions, see Deploy Automatic Failover (if it is configured) on page 19, but do not start the daemon yet.

Step 8: Start services on nn2-alt

Start the NameNode; start the ZKFC for automatic failover; and install and start a JournalNode if you want one to run on nn2-alt. Proceed as follows.
1. Start the JournalNode daemon:

   $ sudo service hadoop-hdfs-journalnode start

2. Start the NameNode daemon:

   $ sudo service hadoop-hdfs-namenode start

3. Start the ZKFC daemon:

   $ sudo service hadoop-hdfs-zkfc start

4. Set these services to restart on boot; for example on a RHEL-compatible system:

   $ sudo chkconfig hadoop-hdfs-namenode on
   $ sudo chkconfig hadoop-hdfs-zkfc on
   $ sudo chkconfig hadoop-hdfs-journalnode on

Step 9: If you are relocating a JournalNode, fail over to nn2-alt

   hdfs haadmin -failover nn1 nn2-alt

Step 10: If you are relocating a JournalNode, restart nn1

   Restart the NameNode daemon on nn1 to force it to re-read the configuration:

   $ sudo service hadoop-hdfs-namenode stop
   $ sudo service hadoop-hdfs-namenode start

Disabling HDFS High Availability

If you need to unconfigure HA and revert to using a single NameNode, either permanently or for upgrade or testing purposes, proceed as follows.

- **Important:**
  
  If you have been using NFS shared storage in CDH 4, you must unconfigure it before upgrading to CDH 5. Only Quorum-based storage is supported in CDH 5. If you already using Quorum-based storage, you do not need to unconfigure it in order to upgrade.

Step 1: Shut Down the Cluster

1. Shut down Hadoop services across your entire cluster. Do this from Cloudera Manager; or, if you are not using Cloudera Manager, run the following command on every host in your cluster:

   $ for x in `cd /etc/init.d ; ls hadoop-*` ; do sudo service $x stop ; done

2. Check each host to make sure that there are no processes running as the hdfs, yarn, mapred or httpfs users from root:

   # ps -aeef | grep java

Step 2: Unconfigure HA

1. Disable the software configuration.
Configuring HDFS High Availability

- If you are using Quorum-based storage and want to unconfigure it, unconfigure the HA properties described under Configuring Software for HDFS HA on page 9.
  
  If you intend to redeploy HDFS HA later, comment out the HA properties rather than deleting them.

- If you were using NFS shared storage in CDH 4, you must unconfigure the properties described below before upgrading to CDH 5.

2. Move the NameNode metadata directories on the standby NameNode. The location of these directories is configured by `dfs.namenode.name.dir` and/or `dfs.namenode.edits.dir`. Move them to a backup location.

Step 3: Restart the Cluster

```bash
for x in `cd /etc/init.d ; ls hadoop-*` ; do sudo service $x start ; done
```

Properties to unconfigure to disable an HDFS HA configuration using NFS shared storage

- **Important:**
  
  HDFS HA with NFS shared storage is not supported in CDH 5. Comment out or delete these properties before attempting to upgrade your cluster to CDH 5. (If you intend to configure HA with Quorum-based storage under CDH 5, you should comment them out rather than deleting them, as they are also used in that configuration.)

Unconfigure the following properties.

- In your `core-site.xml` file:
  
  `fs.defaultFS` (formerly `fs.default.name`)
  
  Optionally, you may have configured the default path for Hadoop clients to use the HA-enabled logical URI. For example, if you used `mycluster` as the NameService ID as shown below, this will be the value of the authority portion of all of your HDFS paths.

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

- In your `hdfs-site.xml` configuration file:
  
  `dfs.nameservices`

```xml
<property>
  <name>dfs.nameservices</name>
  <value>mycluster</value>
</property>
```

- Note:

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

  `dfs.ha.namenodes.[nameservice ID]`
A list of comma-separated NameNode IDs used by DataNodes to determine all the NameNodes in the cluster. For example, if you used mycluster as the NameService ID, and you used nn1 and nn2 as the individual IDs of the NameNodes, you would have configured this as follows:

```xml
<property>
  <name>dfs.ha.namenodes.mycluster</name>
  <value>nn1,nn2</value>
</property>
```

dfs.namenode.rpc-address.[nameservice ID]

For both of the previously-configured NameNode IDs, the full address and RPC port of the NameNode process. For example:

```xml
<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>
```

- **Note:**
  You may have similarly configured the `servicercp-address` setting.

dfs.namenode.http-address.[nameservice ID]

The addresses for both NameNodes’ HTTP servers to listen on. For example:

```xml
<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’s Kerberos security features enabled, and you use HSFTP, you will have set the `https-address` similarly for each NameNode.

dfs.namenode.shared.edits.dir

The path to the remote shared edits directory which the Standby NameNode uses to stay up-to-date with all the file system changes the Active NameNode makes. You should have configured only one of these directories, mounted read/write on both NameNode machines. The value of this setting should be the absolute path to this directory on the NameNode machines. For example:

```xml
<property>
  <name>dfs.namenode.shared.edits.dir</name>
  <value>/mnt/filer1/dfs/ha-name-dir-shared</value>
</property>
```

dfs.client.failover.proxy.provider.[nameservice ID]
The name of the Java class which the DFS Client uses to determine which NameNode is the current Active, and therefore which NameNode is currently serving client requests. The only implementation which shipped with Hadoop is the ConfiguredFailoverProxyProvider. 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 - a list of scripts or Java classes which will be used to fence the Active NameNode during a failover.

- **Note:**
  If you implemented your own custom fencing method, see the org.apache.hadoop.ha.NodeFencer class.

- **The sshfence fencing method**
  sshfence - SSH to the Active NameNode and kill the process
  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 may have configured a non-standard username or port to perform the SSH, as shown below, and also a timeout, in milliseconds, for the SSH:

  ```
  <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>
    <description>SSH connection timeout, in milliseconds, to use with the builtin sshfence fencer.</description>
  </property>
  ```

- **The shell fencing method**
  shell - run an arbitrary shell command to fence the Active NameNode
  The shell fencing method runs an arbitrary shell command, which you may have configured as shown below:

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

**Automatic failover:** If you configured automatic failover, you configured two additional configuration parameters.
In your `hdfs-site.xml`:

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

In your `core-site.xml` file, add:

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

Other properties: There are several other configuration parameters which you may have set to control the behavior of automatic failover, though they were not necessary for most installations. See the configuration section of the Hadoop documentation for details.

Redeploying HDFS High Availability

If you need to redeploy HA using Quorum-based storage after temporarily disabling it, proceed as follows:

1. Shut down the cluster as described in Step 1 of the previous section.
2. Uncomment the properties you commented out in Step 2 of the previous section.
3. Deploy HDFS HA, following the instructions under Deploying HDFS High Availability on page 17.
Configuring MapReduce High Availability (MRv1 or MRv2)

This section covers:

- Configuring High Availability for ResourceManager (MRv2/YARN) on page 31
- Configuring High Availability for the JobTracker (MRv1) on page 37

Configuring High Availability for ResourceManager (MRv2/YARN)

This guide provides an overview of YARN ResourceManager High Availability (RM HA), and explains how to configure and use this feature.

The ResourceManager (RM) is responsible for tracking the resources in a cluster and scheduling applications (for example, MapReduce jobs). Before CDH 5, the RM was a single point of failure in a YARN cluster. The High Availability feature adds redundancy in the form of an Active/Standby ResourceManager pair to remove this single point of failure. Furthermore, upon failover from the Standby ResourceManager to the Active, the applications can resume from their last check-pointed state; for example, completed map tasks in a MapReduce job are not re-run on a subsequent attempt. This allows events such the following to be handled without any significant performance effect on running applications:

- Unplanned events such as machine crashes
- Planned maintenance events such as software or hardware upgrades on the machine running the ResourceManager.

Architecture

ResourceManager HA is implemented by means of an Active/Standby pair of ResourceManagers. On start-up, each RM is in the Standby state: the process is started, but the state is not loaded. When transitioning to active, the RM loads the internal state from the designated state store and starts all the internal services. The stimulus to transition-to-active comes from either the administrator (through the CLI) or through the integrated failover controller when automatic failover is enabled. The subsections that follow provide more details about the components of RM HA.

RM Restart

RM Restart allows restarting the RM, while recovering the in-flight applications if recovery is enabled. To achieve this, the RM stores its internal state, primarily application-related data and tokens, to the RMStateStore; the cluster resources are re-constructed when the NodeManagers connect. The available alternatives for the state store are MemoryRMStateStore (a memory-based implementation), FileSystemRMStateStore (file system-based implementation; HDFS can be used for the file system), and ZKRMStateStore (ZooKeeper-based implementation).

Fencing

When running two RMs, a split-brain situation can arise where both RMs assume they are Active. To avoid this, only a single RM should be able to perform Active operations and the other RM should be "fenced". The ZooKeeper-based state store (ZKRMStateStore) allows a single RM to make changes to the stored state, implicitly fencing the other RM. This is accomplished by the RM claiming exclusive create-delete permissions on the root znode. The ACLs on the root znode are automatically created based on the ACLs configured for the store; in case of secure clusters, Cloudera recommends that you set ACLs for the root node such that both RMs share read-write-admin access, but have exclusive create-delete access. The fencing is implicit and doesn't require explicit configuration (as fencing in HDFS and MRv1 does). You can plug in a custom "Fencer" if you choose to – for example, to use a different implementation of the state store.
Configuring MapReduce High Availability (MRv1 or MRv2)

Configuration and FailoverProxy

In an HA setting, you should configure two RMs to use different ports (for example, ports on different hosts). To facilitate this, YARN uses the notion of an RM Identifier (rm-id). Each RM has a unique rm-id, and all the RPC configurations (<rpc-address>; for example yarn.resourcemanager.address) for that RM can be configured via <rpc-address>.<rm-id>. Clients, ApplicationMasters, and NodeManagers use these RPC addresses to talk to the Active RM automatically, even after a failover. To achieve this, they cycle through the list of RMs in the configuration. This is done automatically and doesn't require any configuration (as in does in HDFS and MRv1).

Manual transitions and failover

You can use the command-line tool yarn rmadmin to transition a particular RM to Active or Standby state, to fail over from one RM to the other, to get the HA state of an RM, and to monitor an RM’s health.

Automatic failover

By default, RM HA uses ZKFC (ZooKeeper-based failover controller) for automatic failover in case the Active RM is unreachable or goes down. Internally, the ActiveStandbyElector is used to elect the Active RM. The failover controller runs as part of the RM (not as a separate process as in HDFS and MapReduce v1) and requires no further setup after the appropriate properties are configured in yarn-site.xml.

You can plug in a custom failover controller if you prefer.

Setting up ResourceManager HA

To configure and start ResourceManager HA, proceed as follows:

1. Stop all YARN daemons,
2. Update the configuration used by the ResourceManagers, NodeManagers and clients
3. Start all YARN daemons

Step 1: Stop the YARN daemons

To stop the YARN daemons:

Stop the MapReduce JobHistory service, ResourceManager service, and NodeManager on all nodes where they are running, as follows:

```
$ sudo service hadoop-mapreduce-historyserver stop
$ sudo service hadoop-yarn-resourcemanager stop
$ sudo service hadoop-yarn-nodemanager stop
```

Step 2: Configure Manual Failover, and Optionally Automatic Failover

To configure failover:

- **Note:**
  Configure the following properties in yarn-site.xml as shown, whether you are configuring manual or automatic failover. They are sufficient to configure manual failover. You need to configure additional properties for automatic failover.

<table>
<thead>
<tr>
<th>Name</th>
<th>Used On</th>
<th>Default Value</th>
<th>Recommended Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yarn.resourcemanager.</td>
<td>ResourceManager,</td>
<td>false</td>
<td>true</td>
<td>Enable HA</td>
</tr>
<tr>
<td>ha.enabled</td>
<td>NodeManager, Client</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

32 | CDH 5 High Availability Guide
<table>
<thead>
<tr>
<th>Name</th>
<th>Used On</th>
<th>Default Value</th>
<th>Recommended Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yarn.resourcemanager.ha.rm-ids</td>
<td>ResourceManager, NodeManager, Client</td>
<td>(None)</td>
<td>Cluster-specific, e.g., rm1,rm2</td>
<td>Comma-separated list of ResourceManager ids in this cluster.</td>
</tr>
<tr>
<td>yarn.resourcemanager.ha.id</td>
<td>ResourceManager</td>
<td>(None)</td>
<td>RM-specific, e.g., e.g., rm1</td>
<td>Id of the current ResourceManager. Must be set explicitly on each ResourceManager to the appropriate value.</td>
</tr>
<tr>
<td>yarn.resourcemanager.address.&lt;rm-id&gt;</td>
<td>ResourceManager, Client</td>
<td>(None)</td>
<td>Cluster-specific</td>
<td>The value of yarn.resourcemanager.address (Client-RM RPC) for this RM. Must be set for all RMs.</td>
</tr>
<tr>
<td>yarn.resourcemanager.scheduler.address.&lt;rm-id&gt;</td>
<td>ResourceManager, Client</td>
<td>(None)</td>
<td>Cluster-specific</td>
<td>The value of yarn.resourcemanager.scheduler.address (AM-RM RPC) for this RM. Must be set for all RMs.</td>
</tr>
<tr>
<td>yarn.resourcemanager.admin.address.&lt;rm-id&gt;</td>
<td>ResourceManager, Client/Admin</td>
<td>(None)</td>
<td>Cluster-specific</td>
<td>The value of yarn.resourcemanager.admin.address (RM administration) for this RM. Must be set for all RMs.</td>
</tr>
<tr>
<td>yarn.resourcemanager.resource-tracker.address.&lt;rm-id&gt;</td>
<td>ResourceManager, NodeManager</td>
<td>(None)</td>
<td>Cluster-specific</td>
<td>The value of yarn.resourcemanager.resource-tracker.address (NM-RM RPC) for this RM. Must be set for all RMs.</td>
</tr>
<tr>
<td>yarn.resourcemanager.webapp.address.&lt;rm-id&gt;</td>
<td>ResourceManager, Client</td>
<td>(None)</td>
<td>Cluster-specific</td>
<td>The value of yarn.resourcemanager.webapp.address (RM webapp) for this RM. Must be set for all RMs.</td>
</tr>
<tr>
<td>yarn.resourcemanager.recovery.enabled</td>
<td>ResourceManager</td>
<td>false</td>
<td>true</td>
<td>Enable job recovery on RM restart or failover.</td>
</tr>
<tr>
<td>yarn.resourcemanager.store.class</td>
<td>ResourceManager</td>
<td>org.apache.hadoop.yarn.server.resourcemanager.recovery.FileSystemRMStateStore</td>
<td>org.apache.hadoop.yarn.server.resourcemanager.recovery.ZKRMStateStore</td>
<td>The RMStateStore implementation to use to store the ResourceManager's internal state. The ZooKeeper-based</td>
</tr>
</tbody>
</table>
### Recommended Value

<table>
<thead>
<tr>
<th>Name</th>
<th>Used On</th>
<th>Default Value</th>
<th>Recommended Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yarn.resourcemanager.zk-address</td>
<td>ResourceManager</td>
<td>(None)</td>
<td>Cluster-specific</td>
<td>The ZooKeeper quorum to use to store the ResourceManager's internal state.</td>
</tr>
<tr>
<td>yarn.resourcemanager.zk-acl</td>
<td>ResourceManager</td>
<td>world:anyone:rwcda</td>
<td>Cluster-specific</td>
<td>The ACLs the ResourceManager uses for the znode structure to store the internal state.</td>
</tr>
<tr>
<td>yarn.resourcemanager.zk-state-store.root-node.acl</td>
<td>ResourceManager</td>
<td>(None)</td>
<td>Cluster-specific</td>
<td>The ACLs used for the root node of the ZooKeeper state store. The ACLs set here should allow both ResourceManagers to read, write, and administer, with exclusive access to create and delete. If nothing is specified, the root node ACLs are automatically generated on the basis of the ACLs specified through <code>yarn.resourcemanager.zk-acl</code>. But that leaves a security hole in a secure setup.</td>
</tr>
</tbody>
</table>

### To configure automatic failover:

Configure the following additional properties in `yarn-site.xml` to configure automatic failover.

<table>
<thead>
<tr>
<th>Name</th>
<th>Used On</th>
<th>Default Value</th>
<th>Recommended Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yarn.resourcemanager.ha.automatic-failover.enabled</td>
<td>ResourceManager</td>
<td>true</td>
<td>true</td>
<td>Enable automatic failover</td>
</tr>
<tr>
<td>yarn.resourcemanager.ha.automatic-failover.embedded</td>
<td>ResourceManager</td>
<td>true</td>
<td>true</td>
<td>Use the <code>EmbeddedElectorService</code> to pick an Active RM from the ensemble</td>
</tr>
</tbody>
</table>
The following is a sample `yarn-site.xml` showing these properties configured:

```xml
<configuration>
<!-- Resource Manager Configs -->
<property>
  <name>yarn.resourcemanager.connect.retry-interval.ms</name>
  <value>2000</value>
</property>
<property>
  <name>yarn.resourcemanager.ha.enabled</name>
  <value>true</value>
</property>
<property>
  <name>yarn.resourcemanager.ha.automatic-failover.enabled</name>
  <value>true</value>
</property>
<property>
  <name>yarn.resourcemanager.ha.automatic-failover.embedded</name>
  <value>true</value>
</property>
<property>
  <name>yarn.resourcemanager.cluster-id</name>
  <value>pseudo-yarn-rm-cluster</value>
</property>
<property>
  <name>yarn.resourcemanager.ha.rm-ids</name>
  <value>rm1,rm2</value>
</property>
<property>
  <name>yarn.resourcemanager.ha.id</name>
  <value>rm1</value>
</property>
<property>
  <name>yarn.resourcemanager.scheduler.class</name>
  <value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler</value>
</property>
<property>
  <name>yarn.resourcemanager.recovery.enabled</name>
  <value>true</value>
</property>
<property>
  <name>yarn.resourcemanager.store.class</name>
  <value>org.apache.hadoop.yarn.server.resourcemanager.recovery.ZKRMStateStore</value>
</property>
<property>
  <name>yarn.resourcemanager.zk.state-store.address</name>
  <value>localhost:2181</value>
</property>
<property>
  <name>yarn.app.mapreduce.am.scheduler.connection.wait.interval-ms</name>
  <value>5000</value>
</property>
</configuration>

<!-- RM1 configs -->
<property>
  <name>yarn.resourcemanager.address.rm1</name>
  <value>host1:23140</value>
</property>
```

### Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Used On</th>
<th>Default Value</th>
<th>Recommended Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yarn.resourcemanager.cluster-id</td>
<td>ResourceManager</td>
<td>No default value.</td>
<td>Cluster-specific</td>
<td>Cluster name used by the ActiveStandbyElector to elect one of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ResourceManagers as leader.</td>
</tr>
</tbody>
</table>

Configuring MapReduce High Availability (MRv1 or MRv2)
### Configuring MapReduce High Availability (MRv1 or MRv2)

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>yarn.resourcemanager.scheduler.address.rm1</td>
<td>host1:23130</td>
</tr>
<tr>
<td>yarn.resourcemanager.webapp.https.address.rm1</td>
<td>host1:23189</td>
</tr>
<tr>
<td>yarn.resourcemanager.webapp.address.rm1</td>
<td>host1:23188</td>
</tr>
<tr>
<td>yarn.resourcemanager.resource-tracker.address.rm1</td>
<td>host1:23125</td>
</tr>
<tr>
<td>yarn.resourcemanager.admin.address.rm1</td>
<td>host1:23141</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>yarn.resourcemanager.address.rm2</td>
<td>host2:23140</td>
</tr>
<tr>
<td>yarn.resourcemanager.scheduler.address.rm2</td>
<td>host2:23130</td>
</tr>
<tr>
<td>yarn.resourcemanager.webapp.https.address.rm2</td>
<td>host2:23189</td>
</tr>
<tr>
<td>yarn.resourcemanager.webapp.address.rm2</td>
<td>host2:23188</td>
</tr>
<tr>
<td>yarn.resourcemanager.resource-tracker.address.rm2</td>
<td>host2:23125</td>
</tr>
<tr>
<td>yarn.resourcemanager.admin.address.rm2</td>
<td>host2:23141</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>yarn.nodemanager.localizer.address</td>
<td>0.0.0.0:23344</td>
</tr>
<tr>
<td>yarn.nodemanager.webapp.address</td>
<td>0.0.0.0:23999</td>
</tr>
<tr>
<td>yarn.nodemanager.aux-services</td>
<td>mapreduce_shuffle</td>
</tr>
<tr>
<td>yarn.nodemanager.local-dirs</td>
<td>/tmp/pseudo-dist/yarn/local</td>
</tr>
<tr>
<td>yarn.nodemanager.log-dirs</td>
<td>/tmp/pseudo-dist/yarn/log</td>
</tr>
<tr>
<td>mapreduce.shuffle.port</td>
<td>23080</td>
</tr>
</tbody>
</table>
Step 3: Re-start the YARN daemons

To re-start the YARN daemons:

Start the MapReduce JobHistory server, ResourceManager, and NodeManager on all nodes where they were previously running, as follows:

$ sudo service hadoop-mapreduce-historyserver start
$ sudo service hadoop-yarn-resourcemanager start
$ sudo service hadoop-yarn-nodemanager start

Using yarn rmadmin to Administer ResourceManager HA

You can use yarn rmadmin on the command line to manage your ResourceManager HA deployment. yarn rmadmin has the following options related to RM HA:

[-transitionToActive <serviceId>]
[-transitionToStandby <serviceId>]
[-getServiceState <serviceId>]
[-checkHealth <serviceId>]
[-help <command>]

where <serviceId> is the rm-id.

Note:

Even though -help lists the -failover option, it is not supported by yarn rmadmin.

Configuring High Availability for the JobTracker (MRv1)

This section describes how to configure JobTracker for high availability. The following topics provide more information and instructions:

- About JobTracker High Availability (HA) on page 37
- Replacing the non-HA JobTracker with the HA JobTracker on page 38
- Configuring JobTracker High Availability on page 40
- Usage Notes on page 49

About JobTracker High Availability (HA)

If you are running MRv1, you can configure the JobTracker to be highly available. You can configure either manual or automatic failover to a warm-standby JobTracker.

Note:

- No equivalent for JobTracker HA is available for YARN at present.
- As with HDFS HA, the JobTracker high availability feature is backward compatible; that is, if you do not want to enable JobTracker high availability, you can simply keep your existing configuration after updating your hadoop-0.20-mapreduce, hadoop-0.20-mapreduce-jobtracker, and hadoop-0.20-mapreduce-tasktracker packages, and start your services as before. You do not need to perform any of the actions described on this page.
To use the high availability feature, you must create a new configuration. This new configuration is designed such that all the nodes in the cluster can have the same configuration; you do not need to deploy different configuration files to different nodes depending on each node’s role in the cluster.

In an HA setup, the `mapred.job.tracker` property is no longer a `host:port` string, but instead specifies a logical name to identify JobTracker instances in the cluster (active and standby). Each distinct JobTracker in the cluster has a different JobTracker ID. To support a single configuration file for all of the JobTrackers, the relevant configuration parameters are suffixed with the JobTracker logical name as well as the JobTracker ID.

The HA JobTracker is packaged separately from the original (non-HA) JobTracker.

**Important:**
You cannot run both HA and non-HA JobTrackers in the same cluster. Do not install the HA JobTracker unless you need a highly available JobTracker. If you install the HA JobTracker and later decide to revert to the non-HA JobTracker, you will need to uninstall the HA JobTracker and re-install the non-HA JobTracker.

Use the sections that follow to install, configure and test JobTracker HA.

**Replacing the non-HA JobTracker with the HA JobTracker**

This section provides instructions for removing the non-HA JobTracker and installing the HA JobTracker.

**Important:**
The HA JobTracker cannot be installed on a node on which the non-HA JobTracker is installed, and vice versa. If the JobTracker is installed, uninstall it following the instructions below before installing the HA JobTracker. Uninstall the non-HA JobTracker whether or not you intend to install the HA JobTracker on the same node.

**Removing the non-HA JobTracker**

You must remove the original (non-HA) JobTracker before you install and run the HA JobTracker. First, you need to stop the JobTracker and TaskTrackers.

**To stop the JobTracker and TaskTrackers:**

1. **Stop the TaskTrackers:** On each TaskTracker system:
   ```
   $ sudo service hadoop-0.20-mapreduce-tasktracker stop
   ```

2. **Stop the JobTracker:** On the JobTracker system:
   ```
   $ sudo service hadoop-0.20-mapreduce-jobtracker stop
   ```

3. **Verify that the JobTracker and TaskTrackers have stopped:**
   ```
   $ ps -eaf | grep -i job
   $ ps -eaf | grep -i task
   ```

**To remove the JobTracker:**

- **On Red Hat-compatible systems:**
  ```
  $ sudo yum remove hadoop-0.20-mapreduce-jobtracker
  ```
• On SLES systems:
  $ sudo zypper remove hadoop-0.20-mapreduce-jobtracker

• On Ubuntu systems:
  sudo apt-get remove hadoop-0.20-mapreduce-jobtracker

Installing the HA JobTracker

Use the following steps to install the HA JobTracker package, and optionally the ZooKeeper failover controller package (needed for automatic failover).

Step 1: Install the HA JobTracker package on two separate nodes

On each JobTracker node:
• On Red Hat-compatible systems:
  $ sudo yum install hadoop-0.20-mapreduce-jobtrackerha

• On SLES systems:
  $ sudo zypper install hadoop-0.20-mapreduce-jobtrackerha

• On Ubuntu systems:
  sudo apt-get install hadoop-0.20-mapreduce-jobtrackerha

Step 2: (Optionally) install the failover controller package

If you intend to enable automatic failover, you need to install the failover controller package.

Note:
The instructions for automatic failover assume that you have set up a ZooKeeper cluster running on three or more nodes, and have verified its correct operation by connecting using the ZooKeeper command-line interface (CLI). See the ZooKeeper documentation for instructions on how to set up a ZooKeeper ensemble.

Install the failover controller package as follows:

On each JobTracker node:
• On Red Hat-compatible systems:
  $ sudo yum install hadoop-0.20-mapreduce-zkfc

• On SLES systems:
  $ sudo zypper install hadoop-0.20-mapreduce-zkfc

• On Ubuntu systems:
  sudo apt-get install hadoop-0.20-mapreduce-zkfc
Configuring MapReduce High Availability (MRv1 or MRv2)

Configuring JobTracker High Availability

Follow the instructions in this section to configure high availability (HA) for the JobTracker.

JobTracker HA reuses the `mapred.job.tracker` parameter in `mapred-site.xml` to identify a JobTracker active-standby pair. In addition, you must enable the existing `mapred.jobtracker.restart.recover`, `mapred.job.tracker.persist.jobstatus.active`, and `mapred.job.tracker.persist.jobstatus.hours` parameters, as well as a number of new parameters, as discussed below.

Configuring and Deploying Manual Failover

Proceed as follows to configure manual failover:

1. Configure the JobTrackers, TaskTrackers, and Clients
2. Start the JobTrackers
3. Activate a JobTracker
4. Verify that failover is working

Step 1: Configure the JobTrackers, TaskTrackers, and clients

Changes to existing configuration parameters

<table>
<thead>
<tr>
<th>Property name</th>
<th>Default</th>
<th>Used on</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapred.job.tracker</td>
<td>local</td>
<td>JobTracker, TaskTracker, client</td>
<td>In an HA setup, the logical name of the JobTracker active-standby pair. In a non-HA setup <code>mapred.job.tracker</code> is a host:port string specifying the JobTracker's RPC address, but in an HA configuration the logical name must not include a port number.</td>
</tr>
<tr>
<td>mapred.jobtracker.restart.recover</td>
<td>false</td>
<td>JobTracker</td>
<td>Whether to recover jobs that were running in the most recent active JobTracker. Must be set to <code>true</code> for JobTracker HA.</td>
</tr>
<tr>
<td>mapred.job.tracker.persist.jobstatus.active</td>
<td>false</td>
<td>JobTracker</td>
<td>Whether to make job status persistent in HDFS. Must be set to <code>true</code> for JobTracker HA.</td>
</tr>
<tr>
<td>mapred.job.tracker.persist.jobstatus.hours</td>
<td>0</td>
<td>JobTracker</td>
<td>The number of hours job status information is retained in HDFS. Must be greater than zero for JobTracker HA.</td>
</tr>
<tr>
<td>mapred.job.tracker.persist.jobstatus.dir</td>
<td>/jobtracker/jobsInfo</td>
<td>JobTracker</td>
<td>The HDFS directory in which job status information is kept persistently. The directory must exist and be owned by the <code>mapred</code> user.</td>
</tr>
</tbody>
</table>
## New configuration parameters

<table>
<thead>
<tr>
<th>Property name</th>
<th>Default</th>
<th>Used on</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapred.jobtrackers.&lt;name&gt;</td>
<td>None</td>
<td>JobTracker, TaskTracker, client</td>
<td>A comma-separated pair of IDs for the active and standby jobtrackers. The <code>&lt;name&gt;</code> is the value of <code>mapred.jobtracker</code>.</td>
</tr>
<tr>
<td>mapred.jobtracker.rpc-address.&lt;name&gt;.&lt;id&gt;</td>
<td>None</td>
<td>JobTracker, TaskTracker, client</td>
<td>The RPC address of an individual JobTracker. <code>&lt;name&gt;</code> refers to the value of <code>mapred.jobtracker</code>; <code>&lt;id&gt;</code> refers to one or other of the values in <code>mapred.jobtrackers.&lt;name&gt;</code>.</td>
</tr>
<tr>
<td>mapred.jobtracker.http.address.&lt;name&gt;.&lt;id&gt;</td>
<td>None</td>
<td>JobTracker, TaskTracker</td>
<td>The HTTP address of an individual JobTracker. (In a non-HA setup <code>mapred.jobtracker.http.address</code> (with no suffix) is the JobTracker’s HTTP address.)</td>
</tr>
<tr>
<td>mapred.ha.jobtracker.rpc-address.&lt;name&gt;.&lt;id&gt;</td>
<td>None</td>
<td>JobTracker, failover controller</td>
<td>The RPC address of the HA service protocol for the JobTracker. The JobTracker listens on a separate port for HA operations which is why this property exists in addition to <code>mapred.jobtracker.rpc-address.&lt;name&gt;.&lt;id&gt;</code>.</td>
</tr>
<tr>
<td>mapred.ha.jobtracker.http-redirect-address.&lt;name&gt;.&lt;id&gt;</td>
<td>None</td>
<td>JobTracker</td>
<td>The HTTP address of an individual JobTracker that should be used for HTTP redirects. The standby JobTracker will redirect all web traffic to the active, and will use this property to discover the URL to use for redirects. A property separate from <code>mapred.jobtracker.http.address.&lt;name&gt;.&lt;id&gt;</code> is needed since the latter may be a wildcard bind address, such as <code>0.0.0.0:50030</code>, which is not suitable for making requests. Note also that <code>mapred.ha.jobtracker.http-redirect-address.&lt;name&gt;.&lt;id&gt;</code> is the HTTP redirect address for the JobTracker with ID <code>&lt;id&gt;</code> for the pair with the logical name <code>&lt;name&gt;</code> - that</td>
</tr>
</tbody>
</table>
### Configuring MapReduce High Availability (MRv1 or MRv2)

<table>
<thead>
<tr>
<th>Property name</th>
<th>Default</th>
<th>Used on</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapred.ha.jobtracker.id</td>
<td>None</td>
<td>JobTracker</td>
<td>The identity of this JobTracker instance. Note that this is optional since each JobTracker can infer its ID from the matching address in one of the <code>mapred.jobtracker.rpc-address.&lt;name&gt;.&lt;id&gt;</code> properties. It is provided for testing purposes.</td>
</tr>
<tr>
<td>mapred.client.failover.proxy.provider.&lt;name&gt;</td>
<td>None</td>
<td>TaskTracker, client</td>
<td>The failover provider class. The only class available is <code>org.apache.hadoop.mapred.ConfiguredFailoverProxyProvider</code>.</td>
</tr>
<tr>
<td>mapred.client.failover.max.attempts</td>
<td>15</td>
<td>TaskTracker, client</td>
<td>The maximum number of times to try to fail over.</td>
</tr>
<tr>
<td>mapred.client.failover.sleep.base.millis</td>
<td>500</td>
<td>TaskTracker, client</td>
<td>The time to wait before the first failover.</td>
</tr>
<tr>
<td>mapred.client.failover.sleep.max.millis</td>
<td>1500</td>
<td>TaskTracker, client</td>
<td>The maximum amount of time to wait between failovers (for exponential backoff).</td>
</tr>
<tr>
<td>mapred.client.failover.connection.retries</td>
<td>0</td>
<td>TaskTracker, client</td>
<td>The maximum number of times to retry between failovers.</td>
</tr>
<tr>
<td>mapred.client.failover.connection.retries.on.timeouts</td>
<td>0</td>
<td>TaskTracker, client</td>
<td>The maximum number of times to retry on timeouts between failovers.</td>
</tr>
<tr>
<td>mapred.ha.fencing.methods</td>
<td>None</td>
<td>failover controller</td>
<td>A list of scripts or Java classes that will be used to fence the active JobTracker during failover. Only one JobTracker should be active at any given time, but you can simply configure <code>mapred.ha.fencing.methods</code> as <code>shell (bin/true)</code> since the JobTrackers fence themselves, and split-brain is avoided by the old active JobTracker shutting itself</td>
</tr>
</tbody>
</table>
**Make changes and additions similar to the following to mapred-site.xml on each node.**

- **Note:** It is simplest to configure all the parameters on all nodes, even though not all of the parameters will be used on any given node. This also makes for robustness if you later change the roles of the nodes in your cluster.

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"/>
<!-- Put site-specific property overrides in this file. -->
<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>logicaljt</value>
    <!-- host:port string is replaced with a logical name -->
  </property>
  <property>
    <name>mapred.jobtrackers.logicaljt</name>
    <value>jt1,jt2</value>
    <description>Comma-separated list of JobTracker IDs.</description>
  </property>
  <property>
    <name>mapred.jobtracker.rpc-address.logicaljt.jt1</name>
    <!-- RPC address for jt1 -->
    <value>myjt1.myco.com:8021</value>
  </property>
  <property>
    <name>mapred.jobtracker.rpc-address.logicaljt.jt2</name>
    <!-- RPC address for jt2 -->
    <value>myjt2.myco.com:8022</value>
  </property>
  <property>
    <name>mapred.job.tracker.http.address.logicaljt.jt1</name>
    <!-- HTTP bind address for jt1 -->
    <value>0.0.0.0:50030</value>
  </property>
  <property>
    <name>mapred.job.tracker.http.address.logicaljt.jt2</name>
    <!-- HTTP bind address for jt2 -->
    <value>0.0.0.0:50031</value>
  </property>
  <property>
    <name>mapred.ha.jobtracker.rpc-address.logicaljt.jt1</name>
    <!-- RPC address for jt1 HA daemon -->
    <value>myjt1.myco.com:8023</value>
  </property>
  <property>
    <name>mapred.ha.jobtracker.rpc-address.logicaljt.jt2</name>
    <!-- RPC address for jt2 HA daemon -->
    <value>myjt2.myco.com:8024</value>
  </property>
  <property>
    <name>mapred.ha.jobtracker.http-redirect-address.logicaljt.jt1</name>  
  </property>
</configuration>
```
Configuring MapReduce High Availability (MRv1 or MRv2)

<!-- HTTP redirect address for jt1 -->
<value>myjt1.myco.com:50030</value>
</property>

<property>
  <name>mapred.ha.jobtracker.http-redirect-address.logicaljt.jt2</name>
  <!-- HTTP redirect address for jt2 -->
  <value>myjt2.myco.com:50031</value>
</property>

<property>
  <name>mapred.jobtracker.restart.recover</name>
  <value>true</value>
</property>

<property>
  <name>mapred.job.tracker.persist.jobstatus.active</name>
  <value>true</value>
</property>

<property>
  <name>mapred.job.tracker.persist.jobstatus.hours</name>
  <value>1</value>
</property>

<property>
  <name>mapred.job.tracker.persist.jobstatus.dir</name>
  <value>/jobtracker/jobsInfo</value>
</property>

<property>
  <name>mapred.client.failover.proxy.provider.logicaljt</name>
  <value>org.apache.hadoop.mapred.ConfiguredFailoverProxyProvider</value>
</property>

<property>
  <name>mapred.client.failover.max.attempts</name>
  <value>15</value>
</property>

<property>
  <name>mapred.client.failover.sleep.base.millis</name>
  <value>500</value>
</property>

<property>
  <name>mapred.client.failover.sleep.max.millis</name>
  <value>1500</value>
</property>

<property>
  <name>mapred.client.failover.connection.retries</name>
  <value>0</value>
</property>

<property>
  <name>mapred.client.failover.connection.retries.on.timeout</name>
  <value>0</value>
</property>

<property>
  <name>mapred.ha.fencing.methods</name>
  <value>shell(/bin/true)</value>
</property>
</configuration>
Step 2: Start the JobTracker daemons

To start the daemons, run the following command on each JobTracker node:

```
$ sudo service hadoop-0.20-mapreduce-jobtrackerha start
```

Step 3: Activate a JobTracker

**Note:**
- You must be the mapred user to use mrhaadmin commands.
- If Kerberos is enabled, do not use `sudo -u mapred` when using the hadoop mrhaadmin command. Instead, you must log in with the mapred Kerberos credentials (the short name must be mapred). See Configuring Hadoop Security in CDH 5 for more information.

Unless automatic failover is configured, both JobTrackers will be in a standby state after the jobtrackerha daemons start up.

If Kerberos is not enabled, use the following commands:

**To find out what state each JobTracker is in:**

```
$ sudo -u mapred hadoop mrhaadmin -getServiceState <id>
```

where `<id>` is one of the values you configured in the mapred.jobtrackers.<name> property – `jt1` or `jt2` in our sample mapred-site.xml files.

**To transition one of the JobTrackers to active and then verify that it is active:**

```
$ sudo -u mapred hadoop mrhaadmin -transitionToActive <id>
$ sudo -u mapred hadoop mrhaadmin -getServiceState <id>
```

where `<id>` is one of the values you configured in the mapred.jobtrackers.<name> property – `jt1` or `jt2` in our sample mapred-site.xml files.

With Kerberos enabled, log in as the mapred user and use the following commands:

**To log in as the mapred user and kinit:**

```
$ sudo su - mapred
$ kinit -kt mapred.keytab mapred/<fully.qualified.domain.name>
```

**To find out what state each JobTracker is in:**

```
$ hadoop mrhaadmin -getServiceState <id>
```

where `<id>` is one of the values you configured in the mapred.jobtrackers.<name> property – `jt1` or `jt2` in our sample mapred-site.xml files.
To transition one of the JobTrackers to active and then verify that it is active:

```bash
$ hadoop mrhaadmin -transitionToActive <id>
$ hadoop mrhaadmin -getServiceState <id>
```

where `<id>` is one of the values you configured in the `mapred.jobtrackers.<name>` property – `jt1` or `jt2` in our sample `mapred-site.xml` files.

Step 4: Verify that failover is working

Use the following commands, depending whether or not Kerberos is enabled.

If Kerberos is not enabled, use the following commands:

To cause a failover from the currently active to the currently inactive JobTracker:

```bash
$ sudo -u mapred hadoop mrhaadmin -failover <id_of_active_JobTracker> <id_of_inactive_JobTracker>
```

For example, if `jt1` is currently active:

```bash
$ sudo -u mapred hadoop mrhaadmin -failover jt1 jt2
```

To verify the failover:

```bash
$ sudo -u mapred hadoop mrhaadmin -getServiceState <id>
```

For example, if `jt2` should now be active:

```bash
$ sudo -u mapred hadoop mrhaadmin -getServiceState jt2
```

With Kerberos enabled, use the following commands:

To log in as the mapred user and kinit:

```bash
$ sudo su - mapred
$ kinit -kt mapred.keytab mapred/<fully.qualified.domain.name>
```

To cause a failover from the currently active to the currently inactive JobTracker:

```bash
$ hadoop mrhaadmin -failover <id_of_active_JobTracker> <id_of_inactive_JobTracker>
```

For example, if `jt1` is currently active:

```bash
$ hadoop mrhaadmin -failover jt1 jt2
```

To verify the failover:

```bash
$ hadoop mrhaadmin -getServiceState <id>
```

For example, if `jt2` should now be active:

```bash
$ hadoop mrhaadmin -getServiceState jt2
```

Configuring and Deploying Automatic Failover

To configure automatic failover, proceed as follows:

1. Configure a ZooKeeper ensemble (if necessary)
2. Configure parameters for manual failover
3. Configure failover controller parameters
4. Initialize the HA state in ZooKeeper
5. Enable automatic failover
6. Verify automatic failover

Step 1: Configure a ZooKeeper ensemble (if necessary)

To support automatic failover you need to set up a ZooKeeper ensemble running on three or more nodes, and verify its correct operation by connecting using the ZooKeeper command-line interface (CLI). See the ZooKeeper documentation for instructions on how to set up a ZooKeeper ensemble.

Note:
If you are already using a ZooKeeper ensemble for automatic NameNode failover, use the same ensemble for automatic JobTracker failover.

Step 2: Configure the parameters for manual failover

See the instructions for configuring the TaskTrackers and JobTrackers under Configuring and Deploying Manual Failover.

Step 3: Configure failover controller parameters

Use the following additional parameters to configure a failover controller for each JobTracker. The failover controller daemons run on the JobTracker nodes.

New configuration parameters

<table>
<thead>
<tr>
<th>Property name</th>
<th>Default</th>
<th>Configure on</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapred.ha.automatic-failover.enabled</td>
<td>false</td>
<td>failover controller</td>
<td>Set to true to enable automatic failover.</td>
</tr>
<tr>
<td>mapred.ha.zkfc.port</td>
<td>8019</td>
<td>failover controller</td>
<td>The ZooKeeper failover controller port.</td>
</tr>
<tr>
<td>ha.zookeeper.quorum</td>
<td>None</td>
<td>failover controller</td>
<td>The ZooKeeper quorum (ensemble) to use for MRZKFailoverController.</td>
</tr>
</tbody>
</table>

Add the following configuration information to mapred-site.xml:

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

<property>
  <name>mapred.ha.zkfc.port</name>
  <value>8018</value>
</property>
```

Add an entry similar to the following to core-site.xml:

```xml
<property>
  <name>ha.zookeeper.quorum</name>
  <value>zk1.example.com:2181,zk2.example.com:2181,zk3.example.com:2181</value>
</property>
```
Step 4: Initialize the HA State in ZooKeeper

After you have configured the failover controllers, the next step is to initialize the required state in ZooKeeper. You can do so by running one of the following commands from one of the JobTracker nodes.

- **Note:**
  The ZooKeeper ensemble must be running when you use this command; otherwise it will not work properly.

```
$ sudo service hadoop-0.20-mapreduce-zkfc init
```

or

```
$ sudo -u mapred hadoop mrzkfc -formatZK
```

This will create a znode in ZooKeeper in which the automatic failover system stores its data.

- **Note:**
  If you are running a secure cluster, see also [Securing access to ZooKeeper](#).

Step 5: Enable automatic failover

To enable automatic failover once you have completed the configuration steps, you need only start the jobtrackerha and zkfc daemons.

**To start the daemons, run the following commands on each JobTracker node:**

```
$ sudo service hadoop-0.20-mapreduce-zkfc start
$ sudo service hadoop-0.20-mapreduce-jobtrackerha start
```

One of the JobTrackers will automatically transition to active.

Step 6: Verify automatic failover

After enabling automatic failover, you should test its operation. To do so, first locate the active JobTracker. To find out what state each JobTracker is in, use the following command:

- **Note:**
  You must be the mapred user to use mrhaadmin commands.

```
$ sudo -u mapred hadoop mrhaadmin -getServiceState <id>
```

where `<id>` is one of the values you configured in the mapred.jobtrackers.<name> property – `jt1` or `jt2` in our sample `mapred-site.xml` files.

Once you have located your active JobTracker, you can cause a failure on that node. For example, you can use `kill -9 <pid of JobTracker>` to simulate a JVM crash. Or you can power-cycle the machine or its network.
interface to simulate different kinds of outages. After you trigger the outage you want to test, the other JobTracker should automatically become active within several seconds. The amount of time required to detect a failure and trigger a failover depends on the configuration of `ha.zookeeper.session-timeout.ms`, but defaults to 5 seconds.

If the test does not succeed, you may have a misconfiguration. Check the logs for the `zkfc` and `jobtrackerha` daemons to diagnose the problem.

Usage Notes

Using the JobTracker Web UI

To use the JobTracker Web UI, use the HTTP address of either JobTracker (that is, the value of `mapred.job.tracker.http.address.<name>.<id>` for either the active or the standby JobTracker). Note the following:

- If you use the URL of the standby JobTracker, you will be redirected to the active JobTracker.
- If you use the URL of a JobTracker that is down, you will not be redirected - you will simply get a "Not Found" error from your browser.

Turning off Job Recovery

After a failover, the newly active JobTracker by default restarts all jobs that were running when the failover occurred. For Sqoop 1 and HBase jobs, this is undesirable because they are not idempotent (that is, they do not behave the same way on repeated execution). For these jobs you should consider setting `mapred.job.restart.recover` to false in the job configuration (JobConf).
Configuring High Availability for Other CDH Components

This section provides information on configuring High Availability for CDH components independently of HDFS. At present Oozie is the only such component. See also Configuring Other CDH Components to Use HDFS HA on page 20.

About Oozie High Availability

In CDH 5, you can configure multiple active Oozie servers against the same database, providing high availability for Oozie. This is supported in both MRv1 or MRv2 (YARN). You need:

- Multiple Oozie servers, preferably identically configured;
- A database that supports multiple concurrent connections, and (preferably) has HA support;
- A ZooKeeper ensemble comprising at least three ZooKeeper servers;
- A loadbalancer (preferably with HA support, for example HAProxy), Virtual IP, or Round-Robin DNS, to provide a single entry-point for users and for callbacks from the Application Master or JobTracker.

For more information, and installation and configuration instructions, see http://archive.cloudera.com/cdh5/cdh/5/oozie.

For instructions on how to set up Oozie HA with Kerberos see Configuring Oozie HA with Kerberos.

Configuring High Availability for Llama

Llama High Availability (HA) uses an Active/Standby architecture, in which the active Llama is automatically elected using the ZooKeeper-based ActiveStandbyElector. The active Llama accepts RPC/Thrift connections and communicates with YARN. The standby Llama monitors the leader information in ZooKeeper, but doesn't accept RPC/Thrift connections.

Fencing

Only one of the Llamas should be active to ensure the resources are not partitioned. Llama uses ZooKeeper Access Control Lists (ACLs) to claim exclusive ownership of the cluster when transitioning to active, and monitors this ownership periodically. If another Llama takes over, the first one realizes it within this period.

Reclaiming Cluster Resources

To claim resources from YARN, Llama spawns YARN applications and runs unmanaged ApplicationMasters. When a Llama goes down, the resources allocated to all the YARN applications spawned by it are not reclaimed until YARN times out those applications (default timeout is 10 minutes). On Llama failure, these resources are reclaimed by means of a Llama that kills any YARN applications spawned by this pair of Llamas.

Configuring HA

Configure Llama HA by modifying the following configuration properties in /etc/llama/conf/llama-site.xml. There is no need for any additional daemons.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
<th>Recommended</th>
</tr>
</thead>
<tbody>
<tr>
<td>llama.am.cluster.id</td>
<td>Cluster ID of the Llama pair, used to differentiate between different Llamas</td>
<td>llama</td>
<td>[cluster-specific]</td>
</tr>
</tbody>
</table>
### Configuring High Availability for Other CDH Components

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
<th>Recommended</th>
</tr>
</thead>
<tbody>
<tr>
<td>llama.am.ha.enabled*</td>
<td>Whether to enable Llama HA</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>llama.am.ha.zk-quorum*</td>
<td>ZooKeeper quorum to use for leader election and fencing</td>
<td>[cluster-specific]</td>
<td></td>
</tr>
<tr>
<td>llama.am.ha.zk-base</td>
<td>Base znode for leader election and fencing data</td>
<td>/llama</td>
<td>[cluster-specific]</td>
</tr>
<tr>
<td>llama.am.ha.zk-timeout-ms</td>
<td>The session timeout, in milliseconds, for connections to ZooKeeper quorum</td>
<td>10000</td>
<td>10000</td>
</tr>
<tr>
<td>llama.am.ha.zk-acl</td>
<td>ACLs to control access to ZooKeeper</td>
<td>world:anyone:rwcd</td>
<td>[cluster-specific]</td>
</tr>
<tr>
<td>llama.am.ha.zk-auth</td>
<td>Authorization information to go with the ACLs</td>
<td>[cluster-acl-specific]</td>
<td></td>
</tr>
</tbody>
</table>

*Required configurations