cloudera®

# Apache HBase Guide

**Important Notice**


**Cloudera, Inc.**
**395 Page Mill Road**
**Palo Alto, CA 94306**
**info@cloudera.com**
**US: 1-888-789-1488**
**Intl: 1-650-362-0488**
**www.cloudera.com**

**Release Information**

Version: Cloudera Enterprise 6.3.x
Date: September 30, 2021

# Table of Contents

# Apache HBase Guide

Apache HBase is a scalable, distributed, column-oriented datastore. Apache HBase provides real-time read/write random access to very large datasets hosted on HDFS.

## Configuration Settings

HBase has a number of settings that you need to configure. For information, see Configuration Settings for HBase on page 9.

By default, HBase ships configured for standalone mode. In this mode of operation, a single JVM hosts the HBase Master, an HBase RegionServer, and a ZooKeeper quorum peer. HBase stores your data in a location on the local filesystem, rather than using HDFS. Standalone mode is only appropriate for initial testing.

*Pseudo-distributed* mode differs from *standalone* mode in that each of the component processes run in a separate JVM. It differs from *distributed mode* in that each of the separate processes run on the same server, rather than multiple servers in a cluster.

## Managing HBase

You can manage and configure various aspects of HBase using Cloudera Manager. For more information, see Managing HBase on page 50.

## HBase Security

For the most part, securing an HBase cluster is a one-way operation, and moving from a secure to an unsecure configuration should not be attempted without contacting Cloudera support for guidance. For an overview security in HBase, see Managing HBase Security on page 93.

For information about authentication and authorization with HBase, see HBase Authentication and Configuring HBase Authorization.

## HBase Replication

If your data is already in an HBase cluster, replication is useful for getting the data into additional HBase clusters. In HBase, cluster replication refers to keeping one cluster state synchronized with that of another cluster, using the write-ahead log (WAL) of the source cluster to propagate the changes. Replication is enabled at column family granularity. Before enabling replication for a column family, create the table and all column families to be replicated, on the destination cluster.

Cluster replication uses an active-push methodology. An HBase cluster can be a source (also called *active*, meaning that it writes new data), a destination (also called *passive*, meaning that it receives data using replication), or can fulfill both roles at once. Replication is asynchronous, and the goal of replication is consistency.

When data is replicated from one cluster to another, the original source of the data is tracked with a cluster ID, which is part of the metadata. In CDH 6, all clusters that have already consumed the data are also tracked. This prevents replication loops.

For more information about replication in HBase, see HBase Replication.

## HBase High Availability

Most aspects of HBase are highly available in a standard configuration. A cluster typically consists of one Master and three or more RegionServers, with data stored in HDFS. To ensure that every component is highly available, configure one or more backup Masters. The backup Masters run on other hosts than the active Master.

For information about configuring high availability in HBase, see HBase High Availability.

## Troubleshooting HBase

The Cloudera HBase packages have been configured to place logs in `/var/log/hbase`. Cloudera recommends tailing the `.log` files in this directory when you start HBase to check for any error messages or failures.

For information about HBase troubleshooting, see Troubleshooting HBase on page 99.

## Upstream Information for HBase

More HBase information is available on the Apache Software Foundation site on the HBase project page.

For Apache HBase documentation, see the following:

- Apache HBase Reference Guide
- Apache HBase API Guide
- Apache HBase Blogs

Because Cloudera does not support all upstream HBase features, always check the Apache HBase documentation against the current version and supported features of HBase included in this version of the CDH distribution.

HBase has its own JIRA issue tracker.

# Configuration Settings for HBase

This section contains information on configuring the Linux host and HDFS for HBase.

## Using DNS with HBase

HBase uses the local hostname to report its IP address. Both forward and reverse DNS resolving should work. If your server has multiple interfaces, HBase uses the interface that the primary hostname resolves to. If this is insufficient, you can set `hbase.regionserver.dns.interface` in the `hbase-site.xml` file to indicate the primary interface. To work properly, this setting requires that your cluster configuration is consistent and every host has the same network interface configuration. As an alternative, you can set `hbase.regionserver.dns.nameserver` in the `hbase-site.xml` file to use a different DNS name server than the system-wide default.

## Using the Network Time Protocol (NTP) with HBase

The clocks on cluster members must be synchronized for your cluster to function correctly. Some skew is tolerable, but excessive skew could generate odd behaviors. Run NTP or another clock synchronization mechanism on your cluster. If you experience problems querying data or unusual cluster operations, verify the system time. For more information about NTP, see the NTP website.

## Setting User Limits for HBase

Because HBase is a database, it opens many files at the same time. The default setting of 1024 for the maximum number of open files on most Unix-like systems is insufficient. Any significant amount of loading will result in failures and cause error message such as `java.io.IOException...(Too many open files)` to be logged in the HBase or HDFS log files. For more information about this issue, see the Apache HBase Book. You may also notice errors such as:

```
2010-04-06 03:04:37,542 INFO org.apache.hadoop.hdfs.DFSClient: Exception
increateBlockOutputStream java.io.EOFException
2010-04-06 03:04:37,542 INFO org.apache.hadoop.hdfs.DFSClient: Abandoning block
blk_-6935524980745310745_1391901
```

Another setting you should configure is the number of processes a user is permitted to start. The default number of processes is typically 1024. Consider raising this value if you experience `OutOfMemoryException` errors.

### Configuring ulimit for HBase Using Cloudera Manager

**Minimum Required Role: Configurator** (also provided by **Cluster Administrator, Full Administrator**)

1. Go to the HBase service.
2. Click the **Configuration** tab.
3. Select **Scope** > **Master** or **Scope** > **RegionServer**.
4. Locate the **Maximum Process File Descriptors** property or search for it by typing its name in the Search box.
5. Edit the property value.

   To apply this configuration property to other role groups as needed, edit the value for the appropriate role group. See Modifying Configuration Properties Using Cloudera Manager.
6. Enter a **Reason for change**, and then click **Save Changes** to commit the changes.
7. Restart the role.
8. Restart the service.

### Configuring ulimit for HBase Using the Command Line

**Important:**

- Follow these command-line instructions on systems that do not use Cloudera Manager.
- This information applies specifically to CDH 6.3.x. See Cloudera Documentation for information specific to other releases.

Cloudera recommends increasing the maximum number of file handles to more than 10,000. Increasing the file handles for the user running the HBase process is an operating system configuration, not an HBase configuration. A common mistake is to increase the number of file handles for a particular user when HBase is running as a different user. HBase prints the ulimit it is using on the first line in the logs. Make sure that it is correct.

To change the maximum number of open files for a user, use the `ulimit -n` command while logged in as that user.

To set the maximum number of processes a user can start, use the `ulimit -u` command. You can also use the `ulimit` command to set many other limits. For more information, see the online documentation for your operating system, or the output of the `man ulimit` command.

To make the changes persistent, add the command to the user's Bash initialization file (typically `~/.bash_profile` or `~/.bashrc` ). Alternatively, you can configure the settings in the `Pluggable Authentication Module (PAM)` configuration files if your operating system uses PAM and includes the `pam_limits.so` shared library.

### Configuring ulimit using Pluggable Authentication Modules Using the Command Line

**Important:**

- Follow these command-line instructions on systems that do not use Cloudera Manager.
- This information applies specifically to CDH 6.3.x. See Cloudera Documentation for information specific to other releases.

If you are using `ulimit`, you must make the following configuration changes:

1. In the `/etc/security/limits.conf` file, add the following lines, adjusting the values as appropriate. This assumes that your HDFS user is called `hdfs` and your HBase user is called `hbase`.

```
hdfs   -       nofile  32768
hdfs   -       nproc   2048
hbase  -       nofile  32768
hbase  -       nproc   2048
```

**Note:**

- Only the `root` user can edit this file.
- If this change does not take effect, check other configuration files in the `/etc/security/limits.d/` directory for lines containing the `hdfs` or `hbase` user and the `nofile` value. Such entries may be overriding the entries in `/etc/security/limits.conf`.

To apply the changes in `/etc/security/limits.conf` on Ubuntu and Debian systems, add the following line in the `/etc/pam.d/common-session` file:

```
session required  pam_limits.so
```

For more information on the `ulimit` command or per-user operating system limits, refer to the documentation for your operating system.

## Using `dfs.datanode.max.transfer.threads` with HBase

A Hadoop HDFS DataNode has an upper bound on the number of files that it can serve at any one time. The upper bound is controlled by the `dfs.datanode.max.transfer.threads` property (the property is spelled in the code exactly as shown here). Before loading, make sure you have configured the value for `dfs.datanode.max.transfer.threads` in the `conf/hdfs-site.xml` file (by default found in `/etc/hadoop/conf/hdfs-site.xml`) to at least `4096` as shown below:

```
<property>
  <name>dfs.datanode.max.transfer.threads</name>
  <value>4096</value>
</property>
```

Restart HDFS after changing the value for `dfs.datanode.max.transfer.threads`. If the value is not set to an appropriate value, strange failures can occur and an error message about exceeding the number of transfer threads will be added to the DataNode logs. Other error messages about missing blocks are also logged, such as:

```
06/12/14 20:10:31 INFO hdfs.DFSClient: Could not obtain block
blk_XXXXXXXXXXXXXXXXXXXXXX_YYYYYYYY from any node:
java.io.IOException: No live nodes contain current block. Will get new block locations
 from namenode and retry...
```

**Note:** The property `dfs.datanode.max.transfer.threads` is a HDFS 2 property which replaces the deprecated property `dfs.datanode.max.xcievers`.

## Configuring BucketCache in HBase

The default `BlockCache` implementation in HBase is `CombinedBlockCache`, and the default off-heap `BlockCache` is `BucketCache`. `SlabCache` is now deprecated. See Configuring the HBase BlockCache on page 22 for information about configuring the `BlockCache` using Cloudera Manager or the command line.

## Configuring Encryption in HBase

It is possible to encrypt the HBase root directory within HDFS, using HDFS Transparent Encryption. This provides an additional layer of protection in case the HDFS filesystem is compromised.

If you use this feature in combination with bulk-loading of HFiles, you must configure `hbase.bulkload.staging.dir` to point to a location within the same encryption zone as the HBase root directory. Otherwise, you may encounter errors such as:

```
org.apache.hadoop.ipc.RemoteException(java.io.IOException):
/tmp/output/f/5237a8430561409bb641507f0c531448 can't be moved into an encryption zone.
```

You can also choose to only encrypt specific column families, which encrypts individual HFiles while leaving others unencrypted, using HBase Transparent Encryption at Rest. This provides a balance of data security and performance.

## Enabling Hedged Reads for HBase

**Minimum Required Role: Configurator** (also provided by **Cluster Administrator, Full Administrator**)

1. Go to the HBase service.
2. Click the **Configuration** tab.
3. Select **Scope** > **HBASE-1 (Service-Wide)**.
4. Select **Category** > **Performance**.

5. Configure the **HDFS Hedged Read Threadpool Size** and **HDFS Hedged Read Delay Threshold** properties. The descriptions for each of these properties on the configuration pages provide more information.
6. Enter a **Reason for change**, and then click **Save Changes** to commit the changes.

## Accessing HBase by using the HBase Shell

After you have started HBase, you can access the database in an interactive way by using the HBase Shell, which is a command interpreter for HBase which is written in Ruby. Always run HBase administrative commands such as the HBase Shell, hbck, or bulk-load commands as the HBase user (typically hbase).

```
hbase shell
```

### HBase Shell Overview

- To get help and to see all available commands, use the `help` command.
- To get help on a specific command, use `help "command"`. For example:

```
hbase> help "create"
```

- To remove an attribute from a table or column family or reset it to its default value, set its value to `nil`. For example, use the following command to remove the `KEEP_DELETED_CELLS` attribute from the `f1` column of the `users` table:

```
hbase> alter 'users', { NAME => 'f1', KEEP_DELETED_CELLS => nil }
```

- To exit the HBase Shell, type `quit`.

### Setting Virtual Machine Options for HBase Shell

HBase in CDH 5.2 and higher allows you to set variables for the virtual machine running HBase Shell, by using the HBASE_SHELL_OPTS environment variable. This example sets several options in the virtual machine.

```
$ HBASE_SHELL_OPTS="-verbose:gc -XX:+PrintGCApplicationStoppedTime -XX:+PrintGCDateStamps
      -XX:+PrintGCDetails -Xloggc:$HBASE_HOME/logs/gc-hbase.log" ./bin/hbase shell
```

### Scripting with HBase Shell

CDH 5.2 and higher include non-interactive mode. This mode allows you to use HBase Shell in scripts, and allow the script to access the exit status of the HBase Shell commands. To invoke non-interactive mode, use the `-n` or `--non-interactive` switch. This small example script shows how to use HBase Shell in a Bash script.

```
#!/bin/bash
echo 'list' | hbase shell -n
status=$?
if [ $status -ne 0 ]; then
  echo "The command may have failed."
fi
```

Successful HBase Shell commands return an exit status of 0. However, an exit status other than 0 does not necessarily indicate a failure, but should be interpreted as unknown. For example, a command may succeed, but while waiting for the response, the client may lose connectivity. In that case, the client has no way to know the outcome of the command. In the case of a non-zero exit status, your script should check to be sure the command actually failed before taking further action.

CDH 5.7 and higher include the `get_splits` command, which returns the split points for a given table:

```
hbase> get_splits 't2'
Total number of splits = 5

=> ["", "10", "20", "30", "40"]
```

You can also write Ruby scripts for use with HBase Shell. Example Ruby scripts are included in the `hbase-examples/src/main/ruby/` directory.

## HBase Online Merge

CDH 6 supports online merging of regions. HBase splits big regions automatically but does not support merging small regions automatically. To complete an online merge of two regions of a table, use the HBase shell to issue the online merge command. By default, both regions to be merged should be neighbors; that is, one end key of a region should be the start key of the other region. Although you can "force merge" any two regions of the same table, this can create overlaps and is not recommended.

The Master and RegionServer both participate in online merges. When the request to merge is sent to the Master, the Master moves the regions to be merged to the same RegionServer, usually the one where the region with the higher load resides. The Master then requests the RegionServer to merge the two regions. The RegionServer processes this request locally. Once the two regions are merged, the new region will be online and available for server requests, and the old regions are taken offline.

For merging two consecutive regions use the following command:

```
hbase> merge_region 'ENCODED_REGIONNAME', 'ENCODED_REGIONNAME'
```

For merging regions that are not adjacent, passing `true` as the third parameter forces the merge.

```
hbase> merge_region 'ENCODED_REGIONNAME', 'ENCODED_REGIONNAME', true
```

**Note:** This command is slightly different from other region operations. You must pass the encoded region name (`ENCODED_REGIONNAME`), not the full region name . The encoded region name is the hash suffix on region names. For example, if the region name is `TestTable,0094429456,1289497600452.527db22f95c8a9e0116f0cc13c680396`, the encoded region name portion is `527db22f95c8a9e0116f0cc13c680396`.

## Configuring RegionServer Grouping

You can use RegionServer Grouping (rsgroup) to impose strict isolation between RegionServers by partitioning RegionServers into distinct groups. You can use HBase Shell commands to define and manage RegionServer Grouping.

You must first create an rsgroup before you can add RegionServers to it. Once you have created an rsgroup, you can move your HBase tables into this rsgroup so that only the RegionServers in the same rsgroup can host the regions of the table.

**Note:** RegionServers and tables can only belong to one rsgroup at a time. By default, all the tables and RegionServers belong to the `default` rsgroup.

A custom balancer implementation tracks assignments per rsgroup and moves regions to the relevant RegionServers in that rsgroup. The rsgroup information is stored in a regular HBase table, and a ZooKeeper-based read-only cache is used at cluster bootstrap time.

### Enabling RegionServer Grouping using Cloudera Manager

**Minimum Required Role: Configurator** (also provided by **Cluster Administrator, Full Administrator**)

You must use Cloudera Manager to enable RegionServer Grouping before you can define and manage rsgroups.

1. Go to the HBase service.
2. Click the **Configuration** tab.
3. Select **Scope** > **Master**.
4. Locate the **HBase Coprocessor Master Classes** property or search for it by typing its name in the Search box.
5. Add the following property value: `org.apache.hadoop.hbase.rsgroup.RSGroupAdminEndpoint`.
6. Locate the **Master Advanced Configuration Snippet (Safety Valve) for hbase-site.xml** property or search for it by typing its name in the Search box.
7. Click View as XML and add the following property:

```
<property>
  <name>hbase.master.loadbalancer.class</name>
  <value>org.apache.hadoop.hbase.rsgroup.RSGroupBasedLoadBalancer</value>
</property>
```

8. Enter a **Reason for change**, and then click **Save Changes** to commit the changes.
9. Restart the role.
10. Restart the service.

### Configuring RegionServer Grouping

**Minimum Required Role: Configurator** (also provided by **Cluster Administrator, Full Administrator**)

When you add a new rsgroup, you are creating an rsgroup other than the default group. To configure a rsgroup, in the HBase shell:

1. Add an rsgroup:

```
hbase> add_rsgroup 'mygroup'
```

2. Add RegionServers and tables to this rsgroup:

```
hbase> move_servers_tables_rsgroup ‘mygroup’,
['server1:port','server2:port'],['table1','table2']
```

3. Run the `balance_rsgroup` command if the tables are slow to migrate to the group's dedicated server.

> **Note:** The term ***rsgroup*** refers to servers in a cluster with only the hostname and port. It does not make use of the HBase ServerName type identifying RegionServers (hostname + port + start time) to distinguish RegionServer instances.

### Monitor RegionServer Grouping

**Minimum Required Role: Configurator** (also provided by **Cluster Administrator, Full Administrator**)

You can monitor the status of the commands using the Tables tab on the HBase Master UI home page. If you click on a table name, you can see the RegionServers that are deployed.

You must manually align the RegionServers referenced in rsgroups with the actual state of nodes in the cluster that is active and running.

### Removing a RegionServer from RegionServer Grouping

**Minimum Required Role: Configurator** (also provided by **Cluster Administrator, Full Administrator**)

You can remove a RegionServer by moving it to the `default` rsgroup. Edits made using shell commands to all rsgroups, except the `default` rsgroup, are persisted to the system `hbase:rsgroup` table. If an rsgroup references a decommissioned RegionServer, then the rsgroup should be updated to undo the reference.

1. Move the RegionServer to the default rsgroup using the command:

```
hbase> move_servers_rsgroup 'default',['server1:port']
```

2. Check the list of RegionServers in your rsgourp to ensure that that the RegionServer is successfully removed using the command:

```
hbase> get_rsgroup 'mygroup'
```

The default rsgroup's RegionServer list mirrors the current state of the cluster. If you shut down a RegionServer that was part of the `default` rsgroup, and then run the `get_rsgroup 'default'` command to list its content in the shell, the server is no longer listed. If you move the offline server from the non-default rsgroup to `default`, it will not show in the `default` list; the server will just be removed from the list.

### Enabling ACL for RegionServer Grouping

**Minimum Required Role: Full Administrator**

You need to be a Global Admin to manage rsgroups if authorization is enabled.

To enable ACL, add the following to the `hbase-site.xml` file, and then restart your HBase Master server:

```
<property>
  <name>hbase.security.authorization</name>
  <value>true</value>
<property>
```

### Best Practices when using RegionServer Grouping

You must keep in mind the following best practices when using rsgroups:

Isolate System Tables

You can either have a system rsgroup where all the system tables are present or just leave the system tables in `default` rsgroup and have all user-space tables in non-`default` rsgroups.

Handling Dead Nodes

You can have a special rsgroup of dead or questionable nodes to help you keep them without running until the nodes are repaired. Be careful when replacing dead nodes in an rsgroup, and ensure there are enough live nodes before you start moving out the dead nodes. You can move the good live nodes first before moving out the dead nodes.

If you have configured a table to be in a rsgroup, but all the RegionServers in that rsgroup die, the tables become unavailable and you can no longer access those tables.

### Troubleshooting RegionServer Grouping

If you encounter an issue when using rsgroup, check the Master log to see what is causing the issue. If an rsgroup operation is unresponsive, restart the Master.

For example, if you have not enabled the rsgroup coprocessor endpoint in the Master, and run any of the rsgroup shell commands, you will encounter the following error message:

```
ERROR: org.apache.hadoop.hbase.exceptions.UnknownProtocolException: No registered master
 coprocessor service found for name RSGroupAdminService
    at
org.apache.hadoop.hbase.master.MasterRpcServices.execMasterService(MasterRpcServices.java:604)

    at
```

```
org.apache.hadoop.hbase.shaded.protobuf.generated.MasterProtos$MasterService$2.callBlockingMethod(MasterProtos.java)

    at org.apache.hadoop.hbase.ipc.RpcServer.call(RpcServer.java:1140)
    at org.apache.hadoop.hbase.ipc.CallRunner.run(CallRunner.java:133)
    at org.apache.hadoop.hbase.ipc.RpcExecutor$Handler.run(RpcExecutor.java:277)
    at org.apache.hadoop.hbase.ipc.RpcExecutor$Handler.run(RpcExecutor.java:257)
```

### Disabling RegionServer Grouping

When you no longer require rsgroups, you can disable it for your cluster. Removing RegionServer Grouping for a cluster on which it was enabled involves more steps in addition to removing the relevant properties from `hbase-site.xml`. You must ensure that you clean the RegionServer grouping-related metadata so that if the feature is re-enabled in the future, the old metadata will not affect the functioning of the cluster.

To disable RegionServer Grouping:

1. Move all the tables in non-default rsgroups to `default` RegionServer group.

```
#Reassigning table t1 from the non-default group - hbase shell
hbase> move_tables_rsgroup 'default',['t1']
```

2. Move all RegionServers in non-default rsgroups to `default` regionserver group.

```
#Reassigning all the servers in the non-default rsgroup to default - hbase shell
hbase> move_servers_rsgroup
'default',['regionserver1:port','regionserver2:port','regionserver3:port']
```

3. Remove all non-default rsgroups. `default` rsgroup created implicitly does not have to be removed.

```
#removing non-default rsgroup - hbase shell
hbase> remove_rsgroup 'mygroup'
```

4. Remove the changes made in `hbase-site.xml` and restart the cluster.
5. Drop the table `hbase:rsgroup` from HBase.

```
#Through hbase shell drop table hbase:rsgroup
hbase> disable 'hbase:rsgroup'
0 row(s) in 2.6270 seconds
hbase> drop 'hbase:rsgroup'
0 row(s) in 1.2730 seconds
```

6. Remove the znode `rsgroup` from the cluster ZooKeeper using zkCli.sh.

```
#From ZK remove the node /hbase/rsgroup through zkCli.sh
rmr /hbase/rsgroup
```

## Troubleshooting HBase

See Troubleshooting HBase.

## Configuring the BlockCache

See Configuring the HBase BlockCache on page 22.

## Configuring the Scanner Heartbeat

See Configuring the HBase Scanner Heartbeat on page 36.

## Accessing HBase by using the HBase Shell

After you have started HBase, you can access the database in an interactive way by using the HBase Shell, which is a command interpreter for HBase which is written in Ruby. Always run HBase administrative commands such as the HBase Shell, `hbck`, or bulk-load commands as the HBase user (typically `hbase`).

```
hbase shell
```

### HBase Shell Overview

- To get help and to see all available commands, use the `help` command.
- To get help on a specific command, use `help "command"`. For example:

```
hbase> help "create"
```

- To remove an attribute from a table or column family or reset it to its default value, set its value to `nil`. For example, use the following command to remove the `KEEP_DELETED_CELLS` attribute from the `f1` column of the `users` table:

```
hbase> alter 'users', { NAME => 'f1', KEEP_DELETED_CELLS => nil }
```

- To exit the HBase Shell, type `quit`.

### Setting Virtual Machine Options for HBase Shell

HBase in CDH 5.2 and higher allows you to set variables for the virtual machine running HBase Shell, by using the `HBASE_SHELL_OPTS` environment variable. This example sets several options in the virtual machine.

```
$ HBASE_SHELL_OPTS="-verbose:gc -XX:+PrintGCApplicationStoppedTime -XX:+PrintGCDateStamps
      -XX:+PrintGCDetails -Xloggc:$HBASE_HOME/logs/gc-hbase.log" ./bin/hbase shell
```

### Scripting with HBase Shell

CDH 5.2 and higher include non-interactive mode. This mode allows you to use HBase Shell in scripts, and allow the script to access the exit status of the HBase Shell commands. To invoke non-interactive mode, use the `-n` or `--non-interactive` switch. This small example script shows how to use HBase Shell in a Bash script.

```
#!/bin/bash
echo 'list' | hbase shell -n
status=$?
if [ $status -ne 0 ]; then
  echo "The command may have failed."
fi
```

Successful HBase Shell commands return an exit status of 0. However, an exit status other than 0 does not necessarily indicate a failure, but should be interpreted as unknown. For example, a command may succeed, but while waiting for the response, the client may lose connectivity. In that case, the client has no way to know the outcome of the command. In the case of a non-zero exit status, your script should check to be sure the command actually failed before taking further action.

CDH 5.7 and higher include the `get_splits` command, which returns the split points for a given table:

```
hbase> get_splits 't2'
Total number of splits = 5

=> ["", "10", "20", "30", "40"]
```

You can also write Ruby scripts for use with HBase Shell. Example Ruby scripts are included in the `hbase-examples/src/main/ruby/` directory.

## HBase Online Merge

CDH 6 supports online merging of regions. HBase splits big regions automatically but does not support merging small regions automatically. To complete an online merge of two regions of a table, use the HBase shell to issue the online merge command. By default, both regions to be merged should be neighbors; that is, one end key of a region should be the start key of the other region. Although you can "force merge" any two regions of the same table, this can create overlaps and is not recommended.

The Master and RegionServer both participate in online merges. When the request to merge is sent to the Master, the Master moves the regions to be merged to the same RegionServer, usually the one where the region with the higher load resides. The Master then requests the RegionServer to merge the two regions. The RegionServer processes this request locally. Once the two regions are merged, the new region will be online and available for server requests, and the old regions are taken offline.

For merging two consecutive regions use the following command:

```
hbase> merge_region 'ENCODED_REGIONNAME', 'ENCODED_REGIONNAME'
```

For merging regions that are not adjacent, passing `true` as the third parameter forces the merge.

```
hbase> merge_region 'ENCODED_REGIONNAME', 'ENCODED_REGIONNAME', true
```

**Note:** This command is slightly different from other region operations. You must pass the encoded region name (`ENCODED_REGIONNAME`), not the full region name . The encoded region name is the hash suffix on region names. For example, if the region name is `TestTable,0094429456,1289497600452.527db22f95c8a9e0116f0cc13c680396`, the encoded region name portion is `527db22f95c8a9e0116f0cc13c680396`.

## Using MapReduce with HBase

To run MapReduce jobs that use HBase, you need to add the HBase and Zookeeper JAR files to the Hadoop Java classpath. You can do this by adding the following statement to each job:

```
TableMapReduceUtil.addDependencyJars(job);
```

This distributes the JAR files to the cluster along with your job and adds them to the job's classpath, so that you do not need to edit the MapReduce configuration.

When getting an `Configuration` object for a HBase MapReduce job, instantiate it using the `HBaseConfiguration.create()` method.

## Configuring HBase Garbage Collection

**Warning:** Configuring the JVM garbage collection for HBase is an advanced operation. Incorrect configuration can have major performance implications for your cluster. Test any configuration changes carefully.

Garbage collection (memory cleanup) by the JVM can cause HBase clients to experience excessive latency. See Tuning Java Garbage Collection for HBase for a discussion of various garbage collection settings and their impacts on performance.

To tune the garbage collection settings, you pass the relevant parameters to the JVM.

Example configuration values are not recommendations and should not be considered as such. This is not the complete list of configuration options related to garbage collection. See the documentation for your JVM for details on these settings.

**`-XX:+UseG1GC`**

Use the 'G1' garbage collection algorithm. You can tune G1 garbage collection to provide a consistent pause time, which benefits long-term running Java processes such as HBase, NameNode, Solr, and ZooKeeper. For more information about tuning G1, see the Oracle documentation on tuning garbage collection.

**`-XX:MaxGCPauseMillis=`*`value`***

The garbage collection pause time. Set this to the maximum amount of latency your cluster can tolerate while allowing as much garbage collection as possible.

**`-XX:+ParallelRefProcEnabled`**

Enable or disable parallel reference processing by using a + or - symbol before the parameter name.

**`-XX:-ResizePLAB`**

Enable or disable resizing of Promotion Local Allocation Buffers (PLABs) by using a + or - symbol before the parameter name.

**`-XX:ParallelGCThreads=`*`value`***

The number of parallel garbage collection threads to run concurrently.

**`-XX:G1NewSizePercent=`*`value`***

The percent of the heap to be used for garbage collection. If the value is too low, garbage collection is ineffective. If the value is too high, not enough heap is available for other uses by HBase.

## Configure HBase Garbage Collection Using Cloudera Manager

**Minimum Required Role: Full Administrator**

1. Go to the HBase service.
2. Click the **Configuration** tab.
3. Select **Scope** > **RegionServer**.
4. Select **Category** > **Advanced**.
5. Locate the **Java Configuration Options for HBase RegionServer** property or search for it by typing its name in the Search box.
6. Add or modify JVM configuration options.
7. Enter a **Reason for change**, and then click **Save Changes** to commit the changes.
8. Restart the role.

## Using the HBase Garbage Collector with JDK11

G1GC is the default garbage collector when JDK11 is used. It operates well with large heap sizes, at least 16G, and can cause low throughput and OOM errors in a heap constrained application environment.

You can improve the performance of G1GC by increasing the number of concurrent making threads. You can set this number using the `-XX:ConcGCThreads=n` option. By default this option is set to 10% of the available total host CPUs. However in an environment with more CPUs, you can increase this number to add more concurrent marking threads.

Althought G1GC is the default garbage collector, you can use other garbage collector with JDK11. For example, you can enable CMS using the `-XX:+UseConcMarkSweepGC` java option at runtime.

JDK11 uses the JVM unified logging framework. You can print garbage collection information with the following command: `-Xlog:gc*`. The result of this command is a detailed overview during and outside the pauses about garbage collection activity, including the type of collection and time spent in specific phases of the pause. If you want to see both the garbage collection and heap information, use the `-Xlog:gc+heap` option.

### Disabling the `BoundedByteBufferPool`

HBase uses a `BoundedByteBufferPool` to avoid fragmenting the heap. The G1 garbage collector reduces the need to avoid fragmenting the heap in some cases. If you use the G1 garbage collector, you can disable the `BoundedByteBufferPool` in HBase. This can reduce the number of "old generation" items that need to be collected. This configuration is experimental.

To disable the `BoundedByteBufferPool`, set the `hbase.ipc.server.reservoir.enabled` property to `false`.

#### Disable the `BoundedByteBufferPool` Using Cloudera Manager

1. Go to the HBase service.
2. Click the **Configuration** tab.
3. Select **Scope** > **RegionServer**.
4. Select **Category** > **Advanced**.
5. Locate the **HBase Service Advanced Configuration Snippet (Safety Valve) for hbase-site.xml** property, or search for it by typing its name in the Search box.
6. Add the following XML:

```
<property>
  <name>hbase.ipc.server.reservoir.enabled</name>
  <value>false</value>
</property>
```

7. Enter a **Reason for change**, and then click **Save Changes** to commit the changes.
8. Restart the service.

## Configuring the HBase Canary

The HBase canary is an optional service that periodically checks that a RegionServer is alive. This canary is different from the Cloudera Service Monitoring canary and is provided by the HBase service. The HBase canary is disabled by default. After enabling the canary, you can configure several different thresholds and intervals relating to it, as well as exclude certain tables from the canary checks. The canary works on Kerberos-enabled clusters if you have the HBase client configured to use Kerberos.

### Configure the HBase Canary Using Cloudera Manager

**Minimum Required Role: Full Administrator**

1. Go to the HBase service.
2. Click the **Configuration** tab.
3. Select **Scope** > **HBase or HBase Service-Wide**.
4. Select **Category** > **Monitoring**.
5. Locate the **HBase Canary** property or search for it by typing its name in the Search box. Several properties have *Canary* in the property name.
6. Select the checkbox.
7. Review other HBase Canary properties to configure the specific behavior of the canary.

   To apply this configuration property to other role groups as needed, edit the value for the appropriate role group. See Modifying Configuration Properties Using Cloudera Manager.

8. Enter a **Reason for change**, and then click **Save Changes** to commit the changes.
9. Restart the role.
10. Restart the service.

## Configuring the Blocksize for HBase

The blocksize is an important configuration option for HBase. HBase data is stored in one (after a major compaction) or more (possibly before a major compaction) HFiles per column family per region. It determines both of the following:

- The blocksize for a given column family determines the smallest unit of data HBase can read from the column family's HFiles.
- It is also the basic unit of measure cached by a RegionServer in the BlockCache.

The default blocksize is 64 KB. The appropriate blocksize is dependent upon your data and usage patterns. Use the following guidelines to tune the blocksize size, in combination with testing and benchmarking as appropriate.

**Warning:** The default blocksize is appropriate for a wide range of data usage patterns, and tuning the blocksize is an advanced operation. The wrong configuration can negatively impact performance.

- Consider the average key/value size for the column family when tuning the blocksize. You can find the average key/value size using the HFile utility:

```
$ hbase org.apache.hadoop.hbase.io.hfile.HFile -f /path/to/HFILE -m -v
...
Block index size as per heapsize: 296
reader=hdfs://srv1.example.com:9000/path/to/HFILE, \
compression=none, inMemory=false, \
firstKey=US6683275_20040127/mimetype:/1251853756871/Put, \
lastKey=US6684814_20040203/mimetype:/1251864683374/Put, \
avgKeyLen=37, avgValueLen=8, \
entries=1554, length=84447
...
```

- Consider the pattern of reads to the table or column family. For instance, if it is common to scan for 500 rows on various parts of the table, performance might be increased if the blocksize is large enough to encompass 500-1000 rows, so that often, only one read operation on the HFile is required. If your typical scan size is only 3 rows, returning 500-1000 rows would be overkill.

  It is difficult to predict the size of a row before it is written, because the data will be compressed when it is written to the HFile. Perform testing to determine the correct blocksize for your data.

### Configuring the Blocksize for a Column Family

You can configure the blocksize of a column family at table creation or by disabling and altering an existing table. These instructions are valid whether or not you use Cloudera Manager to manage your cluster.

```
hbase> create ‘test_table ,{NAME => ‘test_cf , BLOCKSIZE => '262144'}
hbase> disable 'test_table'
hbase> alter 'test_table', {NAME => 'test_cf', BLOCKSIZE => '524288'}
hbase> enable 'test_table'
```

After changing the blocksize, the HFiles will be rewritten during the next major compaction. To trigger a major compaction, issue the following command in HBase Shell.

```
hbase> major_compact 'test_table'
```

Depending on the size of the table, the major compaction can take some time and have a performance impact while it is running.

### Monitoring Blocksize Metrics

Several metrics are exposed for monitoring the blocksize by monitoring the blockcache itself.

## Configuring the HBase BlockCache

In the default configuration, HBase uses a single on-heap cache. If you configure the off-heap `BucketCache`, the on-heap cache is used for Bloom filters and indexes, and the off-heap `BucketCache` is used to cache data blocks. This is called the **Combined** Blockcache configuration. The Combined `BlockCache` allows you to use a larger in-memory cache while reducing the negative impact of garbage collection in the heap, because HBase manages the `BucketCache` instead of relying on the garbage collector.

### Contents of the BlockCache

To size the `BlockCache` correctly, you need to understand what HBase places into it.

- **Your data:** Each time a Get or Scan operation occurs, the result is added to the BlockCache if it was not already cached there. If you use the BucketCache, data blocks are always cached in the BucketCache.
- **Row keys:** When a value is loaded into the cache, its row key is also cached. This is one reason to make your row keys as small as possible. A larger row key takes up more space in the cache.
- **hbase:meta:** The `hbase:meta` catalog table keeps track of which RegionServer is serving which regions. It can consume several megabytes of cache if you have a large number of regions, and has `in-memory` access priority, which means HBase attempts to keep it in the cache as long as possible.
- **Indexes of HFiles:** HBase stores its data in HDFS in a format called *HFile*. These HFiles contain indexes which allow HBase to seek for data within them without needing to open the entire HFile. The size of an index is a factor of the block size, the size of your row keys, and the amount of data you are storing. For big data sets, the size can exceed 1 GB per RegionServer, although the entire index is unlikely to be in the cache at the same time. If you use the BucketCache, indexes are always cached on-heap.
- **Bloom filters:** If you use Bloom filters, they are stored in the BlockCache. If you use the BucketCache, Bloom filters are always cached on-heap.

The sum of the sizes of these objects is highly dependent on your usage patterns and the characteristics of your data. For this reason, the HBase Web UI and Cloudera Manager each expose several metrics to help you size and tune the BlockCache.

### Deciding Whether To Use the BucketCache

The HBase team has published the results of exhaustive BlockCache testing, which revealed the following guidelines.

- If the result of a Get or Scan typically fits completely in the heap, the default configuration, which uses the on-heap `LruBlockCache`, is the best choice, as the L2 cache will not provide much benefit. If the eviction rate is low, garbage collection can be 50% less than that of the `BucketCache`, and throughput can be at least 20% higher.
- Otherwise, if your cache is experiencing a consistently high eviction rate, use the `BucketCache`, which causes 30-50% of the garbage collection of `LruBlockCache` when the eviction rate is high.
- `BucketCache` using *file mode* on solid-state disks has a better garbage-collection profile but lower throughput than `BucketCache` using *off-heap memory*.

### Bypassing the BlockCache

If the data needed for a specific but atypical operation does not all fit in memory, using the BlockCache can be counter-productive, because data that you are still using may be evicted, or even if other data is not evicted, excess garbage collection can adversely effect performance. For this type of operation, you may decide to bypass the BlockCache. To bypass the BlockCache for a given Scan or Get, use the `setCacheBlocks(false)` method.

In addition, you can prevent a specific column family's contents from being cached, by setting its `BLOCKCACHE` configuration to `false`. Use the following syntax in HBase Shell:

```
hbase> alter 'myTable', CONFIGURATION => {NAME => 'myCF', BLOCKCACHE => 'false'}
```

## Cache Eviction Priorities

Both the on-heap cache and the off-heap `BucketCache` use the same cache priority mechanism to decide which cache objects to evict to make room for new objects. Three levels of block priority allow for scan-resistance and in-memory column families. Objects evicted from the cache are subject to garbage collection.

- **Single access priority:** The first time a block is loaded from HDFS, that block is given single access priority, which means that it will be part of the first group to be considered during evictions. Scanned blocks are more likely to be evicted than blocks that are used more frequently.
- **Multi access priority:** If a block in the single access priority group is accessed again, that block is assigned multi access priority, which moves it to the second group considered during evictions, and is therefore less likely to be evicted.
- **In-memory access priority:** If the block belongs to a column family which is configured with the `in-memory` configuration option, its priority is changed to in memory access priority, regardless of its access pattern. This group is the last group considered during evictions, but is not guaranteed not to be evicted. Catalog tables are configured with in-memory access priority.

  To configure a column family for in-memory access, use the following syntax in HBase Shell:

```
hbase> alter 'myTable', 'myCF', CONFIGURATION => {IN_MEMORY => 'true'}
```

  To use the Java API to configure a column family for in-memory access, use the `HColumnDescriptor.setInMemory(true)` method.

## Sizing the BlockCache

When you use the `LruBlockCache`, the blocks needed to satisfy each read are cached, evicting older cached objects if the `LruBlockCache` is full. The size cached objects for a given read may be significantly larger than the actual result of the read. For instance, if HBase needs to scan through 20 HFile blocks to return a 100 byte result, and the HFile blocksize is 100 KB, the read will add `20 * 100` KB to the `LruBlockCache`.

Because the `LruBlockCache` resides entirely within the Java heap, the amount of which is available to HBase and what percentage of the heap is available to the `LruBlockCache` strongly impact performance. By default, the amount of HBase heap reserved for `LruBlockCache` (`hfile.block.cache.size`) is `.40`, or 40%. To determine the amount of heap available for the `LruBlockCache`, use the following formula. The `0.99` factor allows 1% of heap to be available as a "working area" for evicting items from the cache. If you use the BucketCache, the on-heap LruBlockCache only stores indexes and Bloom filters, and data blocks are cached in the off-heap BucketCache.

```
number of RegionServers * heap size * hfile.block.cache.size * 0.99
```

To tune the size of the `LruBlockCache`, you can add RegionServers or increase the total Java heap on a given RegionServer to increase it, or you can tune `hfile.block.cache.size` to reduce it. Reducing it will cause cache evictions to happen more often, but will reduce the time it takes to perform a cycle of garbage collection. Increasing the heap will cause garbage collection to take longer but happen less frequently.

## About the Off-heap BucketCache

If the BucketCache is enabled, it stores data blocks, leaving the on-heap cache free for storing indexes and Bloom filters. The physical location of the `BucketCache` storage can be either in memory (off-heap) or in a file stored in a fast disk.

- **Off-heap:** This is the default configuration.
- **File-based:** You can use the file-based storage mode to store the `BucketCache` on an SSD or FusionIO device,

You can configure a column family to keep its data blocks in the L1 cache instead of the BucketCache, using the `HColumnDescriptor.cacheDataInL1(true)` method or by using the following syntax in HBase Shell:

```
hbase> alter 'myTable', CONFIGURATION => {CACHE_DATA_IN_L1 => 'true'}}
```

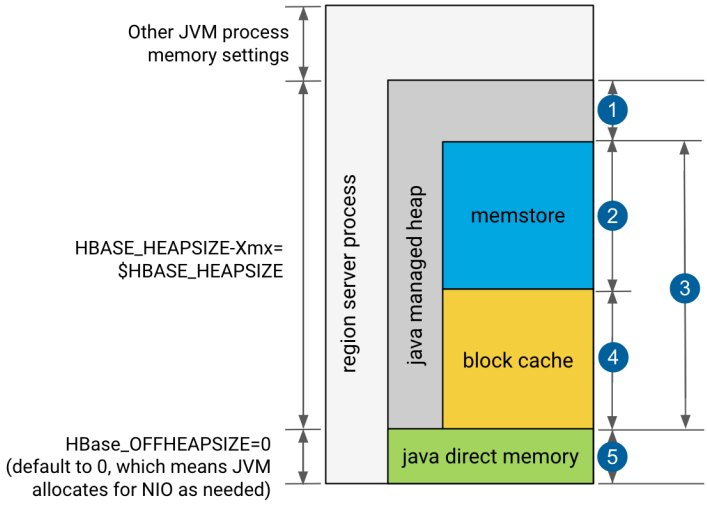
## Configuring the Off-heap BucketCache

This table summaries the important configuration properties for the BucketCache. To configure the BucketCache, see Configuring the Off-heap BucketCache Using Cloudera Manager on page 27 or Configuring the Off-heap BucketCache Using the Command Line on page 28. The table is followed by three diagrams that show the impacts of different blockcache settings.

**Table 1: BucketCache Configuration Properties**

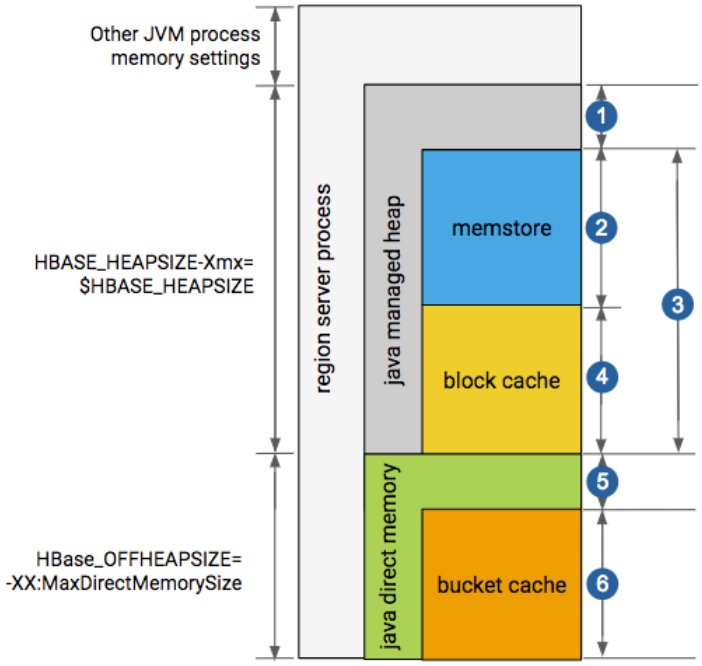
| Property | Default | Description |
|---|---|---|
| hbase.bucketcache.combinedcache.enabled | true | When BucketCache is enabled, use it as a L2 cache for LruBlockCache. If set to true, indexes and Bloom filters are kept in the LruBlockCache and the data blocks are kept in the BucketCache. |
| hbase.bucketcache.ioengine | none (BucketCache is disabled by default) | Where to store the contents of the BucketCache. Its value can be offheap, file:*PATH*, mmap:*PATH* or pmem:*PATH* where PATH is the path to the file that host the file-based cache. |
| hfile.block.cache.size | 0.4 | A float between 0.0 and 1.0. This factor multiplied by the Java heap size is the size of the L1 cache. In other words, the percentage of the Java heap to use for the L1 cache. |
| hbase.bucketcache.size | not set | When using BucketCache, this is a float that represents **one of two different values**, depending on whether it is a floating-point decimal less than 1.0 or an integer greater than 1.0.<br>• If less than 1.0, it represents a percentage of total heap memory size to give to the cache.<br>• If greater than 1.0, it represents the capacity of the cache in megabytes |
| hbase.bucketcache.bucket.sizes | 4, 8, 16, 32, 40, 48, 56, 64, 96, 128, 192, 256, 384, 512 KB | A comma-separated list of sizes for buckets for the BucketCache if you prefer to use multiple sizes. The sizes should be multiples of the default blocksize, ordered from smallest to largest. The sizes you use will depend on your data patterns. **This parameter is experimental.** |
| -XX:MaxDirectMemorySize | MaxDirectMemorySize = BucketCache + 1 | A JVM option to configure the maximum amount of direct memory available for the JVM. It is automatically calculated and configured based on the following |

| Property | Default | Description |
|---|---|---|
| | | formula: `MaxDirectMemorySize = BucketCache size + 1 GB` for other features using direct memory, such as DFSClient. For example, if the BucketCache size is 8 GB, it will be `-XX:MaxDirectMemorySize=9G`. |




1. 20% minimum reserved for operations and rpc call queues
2. `hbase.regionserver.global.memstore.size`: default is 0.4, which means 40%
3. `hbase.regionserver.global.memstore.size` + `hfile.block.cache.size` ≤0.80, which means 80%
4. `hfile.block.cache.size`: default is 0.4, which means 40%
5. slack reserved for HDFS SCR/NIO: number of open HFiles * `hbase.dfs.client.read.shortcircuit.buffer.size`, where `hbase.dfs.client.read.shortcircuit.buffer.size` is set to 128k.

**Figure 1: Default LRUCache, L1 only block cache `hbase.bucketcache.ioengine=NULL`**




1. 20% minimum reserved for operations and rpc call queues
2. `hbase.regionserver.global.memstore.size`: default is 0.4, which means 40%
3. `hbase.regionserver.global.memstore.size` + `hfile.block.cache.size` ≤0.80, which means 80%
4. `hfile.block.cache.size`: default is 0.4 which means 40%
5. slack reserved for HDFS SCR/NIO: number of open HFiles * `hbase.dfs.client.read.shortcircuit.buffer.size`, where `hbase.dfs.client.read.shortcircuit.buffer.size` is set to 128k.
6. `hbase.bucketcache.size`: default is 0.0

   If `hbase.bucketcache.size` is float <1, it represents the percentage of total heap size.

   If `hbase.bucketcache.size` is ≥1, it represents the absolute value in MB. It must be < HBASE_OFFHEAPSIZE

**Figure 2: Default LRUCache, L1 only block cache `hbase.bucketcache.ioengine=offheap`**

### Configuring the BucketCache IO Engine

Use the `hbase.bucketcache.ioengine` parameter to define where to store the content of the `BucketCache`. Its value can be `offheap`, `file:`*PATH*, `mmap:`*PATH*, `pmem:`*PATH* , or it can be empty. By default it is empty which means that BucketCache is disabled.

**Note:** *PATH* denotes the path to the file that host the file-based cache.

**offheap**

When `hbase.bucketcache.ioengine` is set to `offheap`, the content of the BucketCache is stored off-heap as it is presented on the Figure 2: Default LRUCache, L1 only block cache hbase.bucketcache.ioengine=offheap on page 26 image.

**file:PATH**

When `hbase.bucketcache.ioengine` is set to `file:`*PATH*, the BucketCache uses file caching.

**mmap:PATH**

When `hbase.bucketcache.ioengine` is set to `mmap:PATH`, the content of the BucketCache is stored and accessed through memory mapping to a file under the specified path.

**pmem:PATH**

When `hbase.bucketcache.ioengine` is set to `pmem:PATH`, BucketCache uses direct memory access to and from a file on the specified path. The specified path must be under a volume that is mounted on a persistent memory device that supports direct access to its own address space. An example of such persistent memory device is the Intel® Optane™ DC Persistent Memory, when it is mounted in *Direct Mode*.

The advantage of the `pmem` engine over the `mmap` engine is that it supports large cache size. That is because `pmem` allows for reads straight from the device address, which means in this mode no copy is created on DRAM. Therefore, swapping due to DRAM free memory exhaustion is not an issue when large cache size is specified. With devices currently available, the bucket cache size can be set to the order of hundreds of GBs or even a few TBs.

When bucket cache size is set to larger than 256GB, the OS limit must be increased, which can be configured by the `max_map_count` property. Make sure you have an extra 10% for other processes on the host that require the use of memory mapping. This additional overhead depends on the load of processes running on the RS hosts. To calculate the OS limit divide the block cache size in GB by 4 MB and then multiply it by 1.1: `(block cache size in GB / 4 MB) * 1.1`.

Set the value `offheap` and `file:PATH` in the following way:

1. In Cloudera Manager select the HBase service and go to **Configuration**.
2. Search for **BucketCache IOEngine** and set it to the required value.

Set the value `mmap:PATH` and `pmem:PATH` in the following way:

> **Important:** These values can only be set using safety valves.

1. In Cloudera Manager select the HBase service and go to **Configuration**.
2. Search for **RegionServer Advanced Configuration Snippet (Safety Valve) for hbase-site.xml**.
3. Click the plus icon.
4. Set the required value:
   - **Name**: Add `hbase.bucketcache.ioengine`.
   - **Value**: Add either `mmap:PATH:` or `pmem:PATH`.

### Configuring the Off-heap BucketCache Using Cloudera Manager

1. Go to the HBase service.
2. Click the **Configuration** tab.
3. Select the **RegionServer** scope and do the following:
   a. Set **BucketCache IOEngine** to `offheap`.
   b. Update the value of **BucketCache Size** according to the required BucketCache size.
4. In the **RegionServer Environment Advanced Configuration Snippet (Safety Valve)**, edit the `HBASE_REGIONSERVER_OPTS` parameter:

   Add the JVM option `$HBASE_REGIONSERVER_OPTS -XX:MaxDirectMemorySize=<size>G`, replacing `<size>` with a value not smaller than the aggregated heap size expressed as a number of gigabytes + the off-heap BucketCache, expressed as a number of gigabytes + around 1GB used for HDFS short circuit read. For example, if the off-heap BucketCache is 16GB and the heap size is 15GB, the total value of MaxDirectMemorySize could be 32: `-XX:MaxDirectMamorySize=32G`.

If JDK11 is used, Cloudera recommends to replace `<size>` with a value not smaller than the aggregated heap size expressed as a number of gigabytes + 3GB. For example, if the off-heap BucketCache is 16GB, the total value of MaxDirectMemorySize could be 19: `-XX:MaxDirectMamorySize=19G`

```
HBASE_REGIONSERVER_OPTS="$HBASE_REGIONSERVER_OPTS -XX:MaxDirectMemorySize=<size>G"
```

5. Optionally, when combined BucketCache is in use, you can decrease the heap size ratio allocated to the L1 BlockCache, and increase the Memstore size.

   The on-heap BlockCache only stores indexes and Bloom filters, the actual data resides in the off-heap BucketCache. A larger Memstore is able to accommodate more write request before flushing them to disks.

   - Decrease **HFile Block Cache Size** to `0.3` or `0.2`.
   - Increase **Maximum Size of All Memstores in RegionServer** to `0.5` or `0.6` respectively.

6. Enter a **Reason for change**, and then click **Save Changes** to commit the changes.
7. Restart or rolling restart your RegionServers for the changes to take effect.

### Configuring the Off-heap BucketCache Using the Command Line

**Important:**

- Follow these command-line instructions on systems that do not use Cloudera Manager.
- This information applies specifically to CDH 6.3.x. See Cloudera Documentation for information specific to other releases.

1. Configure the MaxDirectMemorySize option for the RegionServers JVMS.

   Add the JVM option `$HBASE_REGIONSERVER_OPTS -XX:MaxDirectMemorySize=<size>G`, replacing `<size>` with a value not smaller than the aggregated heap size expressed as a number of gigabytes + the off-heap BucketCache, expressed as a number of gigabytes + around 1GB used for HDFS short circuit read. For example, if the off-heap BucketCache is 16GB and the heap size is 15GB, the total value of MaxDirectMemorySize could be 32: `-XX:MaxDirectMamorySize=32G`.

   This can be done adding the following line in `hbase-env.sh`:

```
HBASE_REGIONSERVER_OPTS="$HBASE_REGIONSERVER_OPTS -XX:MaxDirectMemorySize=<size>G"
```

2. Next, in the `hbase-site.xml` files on the RegionServers, configure the properties in Table 1: BucketCache Configuration Properties on page 24 as appropriate, using the example below as a model.

```
<property>
  <name>hbase.bucketcache.combinedcache.enabled</name>
  <value>true</value>
</property>
<property>
  <name>hbase.bucketcache.ioengine</name>
  <value>offheap</value>
</property>
<property>
  <name>hbase.bucketcache.size</name>
  <value>8388608</value>
</property>
<property>
  <name>hfile.block.cache.size</name>
  <value>0.2</value>
</property>
<property>
  <name>hbase.regionserver.global.memstore.size</name>
  <value>0.6</value>
</property>
```

Optionally, when combined BucketCache is in use, you can decrease the heap size ratio allocated to the L1 BlockCache, and increase the Memstore size as it is done in the above example. The on-heap BlockCache only stores indexes and Bloom filters, the actual data resides in the off-heap BucketCache. A larger Memstore is able to accommodate more write request before flushing them to disks.

- Decrease `hfile.block.cache.size` to `0.3` or `0.2`.
- Increase `hbase.regionserver.global.memstore.size` to `0.5` or `0.6` respectively.

3. Restart each RegionServer for the changes to take effect.

### Monitoring the BlockCache

Cloudera Manager provides metrics to monitor the performance of the `BlockCache`, to assist you in tuning your configuration.

You can view further detail and graphs using the RegionServer UI. To access the RegionServer UI in Cloudera Manager, go to the Cloudera Manager page for the host, click the **RegionServer** process, and click **HBase RegionServer Web UI**.

If you do not use Cloudera Manager, access the `BlockCache` reports at `http://regionServer_host:22102/rs-status#memoryStats`, replacing *`regionServer_host`* with the hostname or IP address of your RegionServer.

## Configuring Quotas

Two types of HBase quotas are well established: throttle quota and number-of tables-quota. These two quotas can regulate users and tables.

In a multitenant HBase environment, ensuring that each tenant can use only its allotted portion of the system is key in meeting SLAs.

**Table 2: Quota Support Matrix**

| Quota Type | Resource Type | Purpose | Namespace applicable? | Table applicable? | User applicable? |
|---|---|---|---|---|---|
| Throttle | Network | Limit overall network throughput and number of RPC requests | Yes | Yes | Yes |
| New space | Storage | Limit amount of storage used for table or namespaces | Yes | Yes | No |
| Number of tables | Metadata | Limit number of tables for each namespace or user | Yes | No | Yes |
| Numbr of regions | Metadata | Limit number of regions for each namespace | Yes | No | No |

### Setting up quotas

HBase quotas are disabled by default. To enable quotas, the relevant `hbase-site.xml` property must be set to `true` and the limit of each quota specified on the command line.

You need hbase superuser privileges

1. Set the hbase.quota.enabled property in the `hbase-site.xml` file to `true`.
2. Enter the command to set the set the limit of the quota, type of quota, and to which entity to apply the quota. The command and its syntax are:

```
$hbase_shell> set_quota TYPE =>

    quota_type,
    arguments
```

## General Quota Syntax

The general quota syntax are `THROTTLE_TYPE`, `Request sizes and space limit`, `Number of requests`, `Time limits` and `Number of tables or regions`.

**THROTTLE_TYPE**

Can be expressed as READ-only, WRITE-only, or the default type (both READ and WRITE permissions)

**Timeframes**

Can be expressed in the following units of time:

- `sec` (second)

  `min` (minute)

  `hour`

  `day`

**Request sizes and space limit**

Can be expressed in the following units:

- `B`: bytes

  `K`: kilobytes

  `M`: megabytes

  `G`: gigabytes

  `P`: petabytes

When no size units is included, the default value is bytes.

**Number of requests**

Expressed as integer followed by the string request

**Time limits**

Expressed as requests per unit-of-time or size per unit-of-time

*Examples*: `10req/day` or `100P/hour`

**Number of tables or regions**

Expressed as integers

## Throttle quotas

The **throttle quota**, also known as **RPC limit quota**, is commonly used to manage length of RPC queue as well as network bandwidth utilization. It is best used to prioritize time-sensitive applications to ensure latency SLAs are met.

## Throttle quota examples

Following examples details the usage of adding throttle quotas commands, listing throttle quotas commands, and updating and deleting throttle quotas commands.

**Examples of Adding Throttle Quotas Commands**

Limit user u1 to 10 requests per second globally:

```
hbase> set_quota => TYPE => THROTTLE, USER => 'u1', LIMIT => '10req/sec'
```

Limit user u1 to up to 10MB of traffic per second globally:

```
hbase> set_quota => TYPE => THROTTLE, USER => 'u1', LIMIT => '10M/sec'
```

Limit user u1 to 10 requests/second globally for read operations. User u1 can still issue unlimited writes:

```
hbase> set_quota TYPE => THROTTLE, THROTTLE_TYPE => READ, USER => 'u1', LIMIT => '10req/sec'
```

Limit user u1 to 10 requests/second globally for read operations. User u1 can still issue unlimited reads:

```
hbase> set_quota TYPE => THROTTLE, THROTTLE_TYPE => WRITE, USER => 'u1', LIMIT => '10M/sec'
```

Limit user u1 to 5 KB/second for all operations on table t2. User u1 can still issue unlimited requests for other tables, regardless of type of operation:

```
hbase> set_quota TYPE => THROTTLE, USER => 'u1', TABLE => 't2', LIMIT => '5K/min'
```

Limit request to namespaces:

```
hbase> set_quota TYPE => THROTTLE, NAMESPACE => 'ns1', LIMIT => '10req/sec'
```

Limit request to tables:

```
hbase> set_quota TYPE => THROTTLE, TABLE => 't1', LIMIT => '10M/sec'
```

Limit requests based on type, regardless of users, namespaces, or tables:

```
hbase> set_quota TYPE => THROTTLE, THROTTLE_TYPE => WRITE, TABLE => 't1', LIMIT => '10M/sec'
```

**Examples of Listing Throttle Quotas Commands**

Show all quotas:

```
hbase> list_quotas
```

Show all quotas applied to user bob:

```
hbase> list_quotas USER => 'bob.*'
```

Show all quotas applied to user bob and filter by table or namespace:

```
hbase> list_quotas USER => 'bob.*', TABLE => 't1'
hbase> list_quotas USER => 'bob.*', NAMESPACE => 'ns.*'
```

Show all quotas and filter by table or namespace:

```
hbase> list_quotas TABLE => 'myTable'
hbase> list_quotas NAMESPACE => 'ns.*'
```

**Examples of Updating and Deleting Throttle Quotas Commands**

To update a quota, simply issue a new `set_quota` command. To remove a quota, you can set `LIMIT` to `NONE`. The actual quota entry will not be removed, but the policy will be disabled.

```
hbase> set_quota TYPE => THROTTLE, USER => 'u1', LIMIT => NONE
hbase> set_quota TYPE => THROTTLE, USER => 'u1', NAMESPACE => 'ns2', LIMIT => NONE
hbase> set_quota TYPE => THROTTLE, THROTTLE_TYPE => WRITE, USER => 'u1', LIMIT => NONE
hbase> set_quota USER => 'u1', GLOBAL_BYPASS => true
```

## Space quotas

**Space quotas**, also known as **filesystem space quotas**, limit the amount of stored data. It can be applied at a table or namespace level where table-level quotas take priority over namespace-level quotas.

Space quotas are special in that they can trigger different policies when storage goes above thresholds. The following list describes the policies, and they are listed in order of least strict to most strict:

**NO_INSERTS**

Prohibits new data from being ingested (for example, data from `Put`, `Increment`, and `Append` operations are not ingested).

**NO_WRITES**

Performs the same function as `NO_INSERTS` but `Delete` operations are also prohibited.

**NO_WRITES_COMPACTIONS**

Performs the same function as `NO_INSERTS` but compactions are also prohibited.

**DISABLE**

Diables tables.

**Examples of Adding Space Quotas**

Add quota with the condition that Insert operations are rejected when table t1 reaches 1 GB of data:

```
hbase> set_quota TYPE => SPACE, TABLE => 't1', LIMIT => '1G', POLICY => NO_INSERTS
```

Add quota with the condition that table t2 is disabled when 50 GB of data is exceeded:

```
hbase> set_quota TYPE => SPACE, TABLE => 't2', LIMIT => '50G', POLICY => DISABLE
```

Add quota with the condition that Insert and Delete operations cannot be applied to namespace ns1 when it reaches 50 terabytes of data:

```
hbase> set_quota TYPE => SPACE, NAMESPACE => 'ns1', LIMIT => '50T', POLICY => NO_WRITES
```

**Listing Space Quotas**

See "Examples of Listing Throttle Quotas Commands" above for the supported syntax.

**Examples of Updating and Deleting Space Quotas**

A quota can be removed by setting `LIMIT` to `NONE`.

```
hbase> set_quota TYPE => SPACE, TABLE => 't1', LIMIT => NONE
hbase> set_quota TYPE => SPACE, NAMESPACE => 'ns1', LIMIT => NONE
```

## Quota enforcement

When a quota limit is exceeded, the Master server instructs RegionServers to enable an enforcement policy for the namespace or table that violated the quota.

It is important to note the storage quota is not reported in real-time. There is a window when threshold is reached on RegionServers but the threshold accounted for on the Master server is not updated.

**Note:**

Set a storage limit lower than the amount of available disk space to provide extra buffer.

## Quota violation policies

If quotas are set for the amount of space each HBase tenant can fill on HDFS, then a coherent quota violation policy should be planned and implemented on the system.

When a quota violation policy is enabled, the table owner should not be allowed to remove the policy. The expectation is that the Master automatically removes the policy. However, the HBase superuser should still have permission.

Automatic removal of the quota violation policy after the violation is resolved can be accomplished via the same mechanisms that it was originally enforced. But the system should not immediately disable the violation policy when the violation is resolved.

The following describes quota violation policies that you might consider.

**Disabling Tables**

This is the “brute-force” policy, disabling any tables that violated the quota. This policy removes the risk that tables over quota affect your system. For most users, this is likely not a good choice as most sites want READ operations to still succeed.

One hypothetical situation when a disabling tables policy might be advisable is when there are multiple active clusters hosting the same data and, because of a quota violation, it is discovered that one copy of the data does not have all of the data it should have. By disabling tables, you can prevent further discrepancies until the administrator can correct the problem.

**Rejecting All WRITE Operations, Bulk Imports, and Compactions**

This policy rejects all WRITEs and bulk imports to the region which the quota applies. Compactions for this region are also disabled to prevent the system from using more space because of the temporary space demand of a compaction. The only resolution in this case is administrator intervention to increase the quota that is being exceeded.

**Rejecting All WRITE Operations and Bulk Imports**

This is the same as the previous policy, except that compactions are still allowed. This allows users to set or alter a TTL on table and then perform a compaction to reduce the total used space. Inherently, using this violation policy means that you let used space to slightly rise before it is ultimately reduced.

**Allowing DELETE Operations But Rejecting WRITE Operations and Bulk Imports**

This is another variation of the two previously listed policies. This policy allows users to run processes to delete data in the system. Like the previous policy, using this violation policy means that you let used space slightly rises before it is ultimately reduced. In this case, the deletions are propagated to disk and a compaction actually removes data previously stored on disk. TTL configuration and compactions can also be used to remove data.

## Impact of quota violation policy

Quota violation policies can impact live write access, bulk write access, and read access. You must understand what the quota violation policies mean for your deployment before you plan and implement it on your system.

### Live Write Access

As one would expect, every violation policy outlined disables the ability to write new data into the system. This means that any Mutation implementation other than DELETE operations could be rejected by the system. Depending on the violation policy, DELETE operations still might be accepted.

#### Bulk Write Access

Bulk loading HFiles can be an extremely effective way to increase the overall throughput of ingest into HBase. Quota management is very relevant because large HFiles have the potential to quickly violate a quota.

Clients group HFiles by region boundaries and send the file for each column family to the RegionServer presently hosting that region. The RegionServer ultimately inspects each file, ensuring that it should be loaded into this region, and then, sequentially, load each file into the correct column family.

As a part of the precondition-check of the file's boundaries before loading it, the quota state should be inspected to determine if loading the next file will violate the quota. If the RegionServer determines that it will violate the quota, it should not load the file and inform the client that the file was not loaded because it would violate the quota.

### Read Access

In most cases, quota violation policies can affect the ability to read the data stored in HBase. A goal of applying these HBase quotas is to ensure that HDFS remains healthy and sustains a higher level of availability to HBase users. Guaranteeing that there is always free space in HDFS can yield a higher level of health of the physical machines and the DataNodes. This leaves the HDFS-reserved space percentage as a fail-safe mechanism.

### Metrics and Insight

Quotas should ideally be listed on the HBase Master UI. The list of defined quotas should be present as well as those quotas whose violation policy is being enforced. The list of tables/namespaces with enforced violation policies should also be presented via the JMX metrics exposed by the Master.

#### Examples of overlapping quota policies

With the ability to define a quota policy on namespaces and tables, you have to define how the policies are applied. A table quota should take precedence over a namespace quota.

#### Scenario 1

For example, consider Scenario 1, which is outlined in the following table. Namespace *n* has the following collection of tables: *n1.t1*, *n1.t2*, and *n1.t3*. The namespace quota is 100 GB. Because the total storage required for all tables is less than 100 GB, each table can accept new WRITEs.

**Table 3: Scenario 1: Overlapping Quota Policies**

| Object | Quota | Storage Utilization |
|---|---|---|
| Namespace *n1* | 100 GB | 80 GB |
| Table *n1.t1* | 10 GB | 5 GB |
| Table *n1.t2* | (not set) | 50 GB |
| Table *n1.t3* | (not set) | 25 GB |

#### Scenario 2

In Scenario 2, as shown in the following table, WRITEs to table *n1.t1* are denied because the table quota is violated, but WRITEs to table*n1.t2* and table *n1.t3* are still allowed because they are within the namespace quota. The violation policy for the table quota on table *n1.t1* is enacted.

**Table 4: Scenario 2: Overlapping Quota Policies**

| Object | Quota | Storage Utilization |
|---|---|---|
| Namespace *n1* | 100 GB | 60 GB |
| Table *n1.t1* | 10 GB | 15 GB |
| Table *n1.t2* | (not set) | 30 GB |
| Table *n1.t3* | (not set) | 15 GB |

#### Scenario 3

In the Scenario 3 table below, WRITEs to all tables are not allowed because the storage utilization of all tables exceeds the namespace quota limit. The namespace quota violation policy is applied to all tables in the namespace.

**Table 5: Scenario 3: Overlapping Quota Policies**

| Object | Quota | Storage Utilization |
|---|---|---|
| Namespace *n1* | 100 GB | 108 GB |
| Table *n1.t1* | 10 GB | 8 GB |
| Table *n1.t2* | (not set) | 50 GB |
| Table *n1.t3* | (not set) | 50 GB |

### Scenario 4

In the Scenario 4 table below, table *n1.t1* violates the quota set at the table level. The table quota violation policy is enforced. In addition, the disk utilization of table *n1t1* plus the sum of disk utilization for table *n1t2* and table *n1t3*exceeds the 100 GB namespace quota. Therefore, the namespace quota violation policy is also applied.

**Table 6: Scenario 4: Overlapping Quota Policies**

| Object | Quota | Storage Utilization |
|---|---|---|
| Namespace *n1* | 100 GB | 115 GB |
| Table *n1.t1* | 10 GB | 15 GB |
| Table *n1.t2* | (not set) | 50 GB |
| Table *n1.t3* | (not set) | 50 GB |

## Number-of-Tables Quotas

The number-of-tables quota is set as part of the namespace metadata and does not involve the `set_quota` command.

**Examples of Commands Relevant to Setting and Administering Number-of-Tables Quotas**

Create namespace *ns1* with a maximum of 5 tables

```
hbase> create_namespace 'ns1', {'hbase.namespace.quota.maxtables'=>'5'}
```

Alter an existing namespace *ns1* to set a maximum of 8 tables

```
hbase> alter_namespace 'ns1', {METHOD => 'set', 'hbase.namespace.quota.maxtables'=>'8'}
```

Show quota information for namespace *ns1*

```
hbase> describe_namespace 'ns1'
```

Alter existing namespace *ns1* to remove a quota

```
hbase> alter_namespace 'ns1', {METHOD => 'unset', NAME=>'hbase.namespace.quota.maxtables'}
```

## Number-of-Regions Quotas

The number-of-regions quota is similar to the number-of-tables quota. The number-of-regions quota is set as part of the namespace metadata and does not involve the `set_quota` command.

**Examples of Commands Relevant to Setting and Administering Number-of-Regions Quotas**

Create namespace *ns1* with a maximum of 5 tables

```
hbase> create_namespace 'ns1', {'hbase.namespace.quota.maxregions'=>'5'}
```

Alter an existing namespace *ns1* to set a maximum of 8 regions

```
hbase> alter_namespace 'ns1', {METHOD => 'set', 'hbase.namespace.quota.maxregions'=>'8'}
```

Show quota information for namespace *ns1*

```
hbase> describe_namespace 'ns1'
```

Alter existing namespace *ns1* to remove a quota

```
hbase> alter_namespace 'ns1', {METHOD => 'unset', NAME=>'hbase.namespace.quota.maxregions'}
```

## Configuring the HBase Scanner Heartbeat

A *scanner heartbeat check* enforces a time limit on the execution of scan RPC requests. This helps prevent scans from taking too long and causing a timeout at the client.

When the server receives a scan RPC request, a time limit is calculated to be half of the smaller of two values: `hbase.client.scanner.timeout.period` and `hbase.rpc.timeout` (which both default to 60000 milliseconds, or one minute). When the time limit is reached, the server returns the results it has accumulated up to that point. This result set may be empty. If your usage pattern includes that scans will take longer than a minute, you can increase these values.

To make sure the timeout period is not too short, you can configure `hbase.cells.scanned.per.heartbeat.check` to a minimum number of cells that must be scanned before a timeout check occurs. The default value is 10000. A smaller value causes timeout checks to occur more often.

### Configure the Scanner Heartbeat Using Cloudera Manager

**Minimum Required Role: Configurator** (also provided by **Cluster Administrator, Full Administrator**)

1. Go to the HBase service.
2. Click the **Configuration** tab.
3. Select **HBase or HBase Service-Wide**.
4. Locate the **RPC Timeout** property by typing its name in the Search box, and edit the property.
5. Locate the **HBase RegionServer Lease Period** property by typing its name in the Search box, and edit the property.
6. Enter a **Reason for change**, and then click **Save Changes** to commit the changes.
7. Restart the role.
8. Restart the service.

## Limiting the Speed of Compactions

You can limit the speed at which HBase compactions run, by configuring `hbase.regionserver.throughput.controller` and its related settings. The default controller is `org.apache.hadoop.hbase.regionserver.throttle.PressureAwareCompactionThroughputController`, which uses the following algorithm:

1. If compaction pressure is greater than 1.0, there is no speed limitation.
2. In off-peak hours, use a fixed throughput limitation, configured using `hbase.hstore.compaction.throughput.offpeak`, `hbase.offpeak.start.hour`, and `hbase.offpeak.end.hour`.
3. In normal hours, the max throughput is tuned between `hbase.hstore.compaction.throughput.higher.bound` and `hbase.hstore.compaction.throughput.lower.bound` (which default to 20 MB/sec and 10 MB/sec respectively), using the following formula, where `compactionPressure` is between 0.0 and 1.0. The `compactionPressure` refers to the number of store files that require compaction.

```
lower + (higher - lower) * compactionPressure
```

To disable compaction speed limits, set `hbase.regionserver.throughput.controller` to `org.apache.hadoop.hbase.regionserver.throttle.NoLimitThroughputController`.

### Configure the Compaction Speed Using Cloudera Manager

**Minimum Required Role:** **Configurator** (also provided by **Cluster Administrator, Full Administrator**)

1. Go to the HBase service.
2. Click the **Configuration** tab.
3. Select **HBase or HBase Service-Wide**.
4. Search for **HBase Service Advanced Configuration Snippet (Safety Valve) for hbase-site.xml**. Paste the relevant properties from the following example into the field and modify the values as needed:

```
<property>
  <name>hbase.hstore.compaction.throughput.higher.bound</name>
  <value>20971520</value>
  <description>The default is 20 MB/sec</description>
</property>
<property>
  <name>hbase.hstore.compaction.throughput.lower.bound</name>
  <value>10485760</value>
  <description>The default is 10 MB/sec</description>
</property>
<property>
  <name>hbase.hstore.compaction.throughput.offpeak</name>
  <value>9223372036854775807</value>
  <description>The default is Long.MAX_VALUE, which effectively means no
limitation</description>
</property>
<property>
  <name>hbase.offpeak.start.hour</name>
  <value>20</value>
  <description>When to begin using off-peak compaction settings, expressed as an integer
 between 0 and 23.</description>
</property>
<property>
  <name>hbase.offpeak.end.hour</name>
  <value>6</value>
  <description>When to stop using off-peak compaction settings, expressed as an integer
 between 0 and 23.</description>
</property>
<property>
  <name>hbase.hstore.compaction.throughput.tune.period</name>
  <value>60000</value>
</property>
```

5. Enter a **Reason for change**, and then click **Save Changes** to commit the changes.
6. Restart the service.

## Configuring and Using the HBase REST API

You can use the HBase REST API to interact with HBase services, tables, and regions using HTTP endpoints.

### Installing the REST Server

The HBase REST server is an optional component of HBase and is not installed by default.

#### Installing the REST Server Using Cloudera Manager

**Minimum Required Role:** **Full Administrator**

1. Click the **Clusters** tab.
2. Select **Clusters** > ***HBase***.
3. Click the **Instances** tab.
4. Click **Add Role Instance**.
5. Under **HBase REST Server**, click **Select Hosts**.
6. Select one or more hosts to serve the HBase Rest Server role. Click **Continue**.

7. Select the HBase Rest Server roles. Click **Actions For Selected** > **Start**.

## Using the REST API

The HBase REST server exposes endpoints that provide CRUD (create, read, update, delete) operations for each HBase process, as well as tables, regions, and namespaces. For a given endpoint, the HTTP verb controls the type of operation (create, read, update, or delete).

**Note: `curl` Command Examples**

The examples in these tables use the `curl` command, and follow these guidelines:

- The HTTP verb is specified using the `-X` parameter.
- For GET queries, the `Accept` header is set to `text/xml`, which indicates that the client (curl) expects to receive responses formatted in XML. You can set it to `text/json` to receive JSON responses instead.
- For PUT, POST, and DELETE queries, the `Content-Type` header should be set only if data is also being sent with the `-d` parameter. If set, it should match the format of the data being sent, to enable the REST server to deserialize the data correctly.

For more details about the `curl` command, see the documentation for the `curl` version that ships with your operating system.

These examples use port 20050, which is the default port for the HBase REST server when you use Cloudera Manager. If you use CDH without Cloudera Manager, the default port for the REST server is 8080.

**Table 7: Cluster-Wide Endpoints**

| Endpoint | HTTP Verb | Description | Example |
|---|---|---|---|
| /version/cluster | GET | Version of HBase running on this cluster | `curl -vi -X GET \ -H "Accept: text/xml" \ "http://example.com:20550/version/cluster"` |
| /status/cluster | GET | Cluster status | `curl -vi -X GET \ -H "Accept: text/xml" \ "http://example.com:20550/status/cluster"` |
| / | GET | List of all nonsystem tables | `curl -vi -X GET \ -H "Accept: text/xml" \ "http://example.com:20550/"` |

**Table 8: Namespace Endpoints**

| Endpoint | HTTP Verb | Description | Example |
|---|---|---|---|
| /namespaces | GET | List all namespaces. | `curl -vi -X GET \ -H "Accept: text/xml" \ "http://example.com:20550/namespaces/"` |
| /namespaces/*namespace* | GET | Describe a specific namespace. | `curl -vi -X GET \ -H "Accept: text/xml" \ "http://example.com:20550/namespaces/special_ns"` |
| /namespaces/*namespace* | POST | Create a new namespace. | `curl -vi -X POST \ -H "Accept: text/xml" \` |

| Endpoint | HTTP Verb | Description | Example |
|---|---|---|---|
| | | | `"example.com:20550/namespaces/special_ns"` |
| /namespaces/*namespace*/tables | GET | List all tables in a specific namespace. | `curl -vi -X GET \ -H "Accept: text/xml" \ "http://example.com:20550/namespaces/special_ns/tables"` |
| /namespaces/*namespace* | PUT | Alter an existing namespace. Currently not used. | `curl -vi -X PUT \ -H "Accept: text/xml" \ "http://example.com:20550/namespaces/special_ns"` |
| /namespaces/*namespace* | DELETE | Delete a namespace. The namespace must be empty. | `curl -vi -X DELETE \ -H "Accept: text/xml" \ "example.com:20550/namespaces/special_ns"` |

**Table 9: Table Endpoints**

| Endpoint | HTTP Verb | Description | Example |
|---|---|---|---|
| /*table*/schema | GET | Describe the schema of the specified table. | `curl -vi -X GET \ -H "Accept: text/xml" \ "http://example.com:20550/users/schema"` |
| /*table*/schema | POST | Create a new table, or replace an existing table's | `curl -vi -X POST \ -H "Accept: text/xml" \` |

| Endpoint | HTTP Verb | Description | Example |
|---|---|---|---|
| | | schema with the provided schema | `-H "Content-Type: text/xml" \ -d '<?xml version="1.0" encoding="UTF-8"?><TableSchema name="users"><ColumnSchema name="cf" /></TableSchema>' \ "http://example.com:20550/users/schema"` |
| /*table*/schema | UPDATE | Update an existing table with the provided schema fragment | `curl -vi -X PUT \ -H "Accept: text/xml" \ -H "Content-Type: text/xml" \ -d '<?xml version="1.0" encoding="UTF-8"?><TableSchema name="users"><ColumnSchema name="cf" KEEP_DELETED_CELLS="true" /></TableSchema>' \ "http://example.com:20550/users/schema"` |
| /*table*/schema | DELETE | Delete the table. You must use the *table*/schema endpoint, not just *table*/. | `curl -vi -X DELETE \ -H "Accept: text/xml" \ "http://example.com:20550/users/schema"` |
| /*table*/regions | GET | List the table regions. | `curl -vi -X GET \ -H "Accept: text/xml" \ "http://example.com:20550/users/regions"` |

**Table 10: Endpoints for `Get` Operations**

| Endpoint | HTTP Verb | Description | Example |
|---|---|---|---|
| /*table*/*row*/*column:qualifier*/*timestamp* | GET | Get the value of a single row. Values are Base-64 encoded. | Latest version: `curl -vi -X GET \ -H "Accept: text/xml" \ "http://example.com:20550/users/row1"` |

| Endpoint | HTTP Verb | Description | Example |
|---|---|---|---|
| | | | Specific timestamp:<br>`curl -vi -X GET \ -H "Accept: text/xml" \ "http://example.com:20550/users/row1/cf:a/1458586888395"` |
| | | Get the value of a single column. Values are Base-64 encoded. | Latest version:<br>`curl -vi -X GET \ -H "Accept: text/xml" \ "http://example.com:20550/users/row1/cf:a"`<br>Specific version:<br>`curl -vi -X GET \ -H "Accept: text/xml" \ "http://example.com:20550/users/row1/cf:a/"` |
| */table/row/column:qualifier?v=number_of_versions* | | Multi-Get a specified number of versions of a given cell. Values are Base-64 encoded. | `curl -vi -X GET \ -H "Accept: text/xml" \ "http://example.com:20550/users/row1/cf:a?v=2"` |

**Table 11: Endpoints for Scan Operations**

| Endpoint | HTTP Verb | Description | Example |
|---|---|---|---|
| /*table*/scanner/ | PUT | Get a Scanner object. Required by all other Scan operations. Adjust the batch parameter to the number of rows the scan should return in a batch. See the next example for adding filters to your Scanner. The scanner endpoint URL is returned as | `curl -vi -X PUT \ -H "Accept: text/xml" \ -H "Content-Type: text/xml" \ -d '<Scanner batch="1"/>' \` |

| Endpoint | HTTP Verb | Description | Example |
|---|---|---|---|
| | | the `Location` in the HTTP response. The other examples in this table assume that the Scanner endpoint is [illegible] | `"http://example.com:20550/users/scanner/"` |
| /*table*/scanner/ | PUT | To supply filters to the Scanner object or configure the Scanner in any other way, you can create a text file and add your filter to the file. For example, to return only rows for which keys start with `u123` and use a batch size of 100:<br>`<Scanner batch="100"><br><filter><br>{<br>"type": "PrefixFilter",<br>"value": "u123"<br>}<br></filter><br></Scanner>`<br>Pass the file to the `-d` argument of the `curl` request. | `curl -vi -X PUT \ -H "Accept: text/xml" \ -H "Content-Type:text/xml" \ -d @filter.txt \ "http://example.com:20550/users/scanner/"` |
| /*table*/scanner/*scanner_id* | GET | Get the next batch from the scanner. Cell values are byte-encoded. If the scanner | `curl -vi -X GET \ -H "Accept: text/` |

| Endpoint | HTTP Verb | Description | Example |
|---|---|---|---|
| | | is exhausted, HTTP status 204 is returned. | `xml" \ "http://example.com:20550/users/scanner/1458690728243755222O7"` |
| */table*/scanner/*scanner_id* | DELETE | Deletes the scanner and frees the resources it was using. | `curl -vi -X DELETE \ -H "Accept: text/xml" \ "http://example.com:20550/users/scanner/1458690728243755222O7"` |

**Table 12: Endpoints for `Put` Operations**

| Endpoint | HTTP Verb | Description | Example |
|---|---|---|---|
| */table/row_key/* | PUT | Write a row to a table. The row, column qualifier, and value must each be Base-64 encoded. To encode a string, you can use the `base64` command-line utility. To decode the string, use `base64 -d`. The payload is in the --data argument, so the `/users/fakerow` value is a placeholder. Insert multiple rows by adding them to the `<CellSet>` element. You can also save the data to be inserted to a file and pass it to the `-d` parameter with the syntax `-d @filename.txt`. | XML:<br>`curl -vi -X PUT \ -H "Accept: text/xml" \ -H "Content-Type: text/xml" \ -d '<?xml version="1.0" encoding="UTF-8" standalone="yes"?><CellSet><Row key="cm93NQo="><Cell column="Y2ZQo=">dmFsdWU1Cg==</Cell></Row></CellSet>' \ "http://example.com:20550/users/fakerow"`<br>JSON:<br>`curl -vi -X PUT \ -H "Accept: text/json" \ -H "Content-Type: text/json" \ -d '{"Row":[{"key":"cm93NQo=", "Cell": [{"column":"Y2ZQo=", "$":"dmFsdWU1Cg=="}]}]}'' \` |

| Endpoint | HTTP Verb | Description | Example |
|---|---|---|---|
| | | | `"example.com:20550/users/fakerow"` |

## Configuring HBase MultiWAL Support

CDH supports multiple write-ahead logs (MultiWAL) for HBase. (For more information, see HBASE-5699.)

Without MultiWAL support, each region on a RegionServer writes to the same WAL. A busy RegionServer might host several regions, and each write to the WAL is serial because HDFS only supports sequentially written files. This causes the WAL to negatively impact performance.

MultiWAL allows a RegionServer to write multiple WAL streams in parallel by using multiple pipelines in the underlying HDFS instance, which increases total throughput during writes.

**Note:** In the current implementation of MultiWAL, incoming edits are partitioned by Region. Therefore, throughput to a single Region is not increased.

To configure MultiWAL for a RegionServer, set the value of the property `hbase.wal.provider` to `multiwal` and restart the RegionServer. To disable MultiWAL for a RegionServer, unset the property and restart the RegionServer.

RegionServers using the original WAL implementation and those using the MultiWAL implementation can each handle recovery of either set of WALs, so a zero-downtime configuration update is possible through a rolling restart.

### Configuring MultiWAL Support Using Cloudera Manager

1. Go to the HBase service.
2. Click the **Configuration** tab.
3. Select **Scope** > **RegionServer**.
4. Select **Category** > **Main**.
5. Set **WAL Provider** to **MultiWAL**.
6. Set the **Per-RegionServer Number of WAL Pipelines** to a value greater than 1.
7. Enter a **Reason for change**, and then click **Save Changes** to commit the changes.
8. Restart the RegionServer roles.

## Storing Medium Objects (MOBs) in HBase

Data comes in many sizes, and saving all of your data in HBase, including binary data such as images and documents, is convenient. HBase can technically handle binary objects with cells that are up to 10 MB in size. However, HBase normal read and write paths are optimized for values smaller than 100 KB in size. When HBase handles large numbers of values up to 10 MB (medium objects, or MOBs), performance is degraded because of write amplification caused by splits and compactions.

One way to solve this problem is by storing objects larger than 100KB directly in HDFS, and storing references to their locations in HBase. CDH includes optimizations for storing MOBs directly in HBase based on HBASE-11339.

To use MOB, you must use HFile version 3. Optionally, you can configure the MOB file reader's cache settings Service-Wide and for each RegionServer, and then configure specific columns to hold MOB data. No change to client code is required for HBase MOB support.

#### Enabling HFile Version 3 Using Cloudera Manager

**Minimum Required Role: Full Administrator**

To enable HFile version 3 using Cloudera Manager, edit the HBase Service Advanced Configuration Snippet for HBase Service-Wide.

1. Go to the HBase service.
2. Click the **Configuration** tab.
3. Search for the property **HBase Service Advanced Configuration Snippet (Safety Valve)** for `hbase-site.xml`.
4. Paste the following XML into the **Value** field and save your changes.

```
<property>
  <name>hfile.format.version</name>
  <value>3</value>
</property>
```

Changes will take effect after the next major compaction.

## Configuring Columns to Store MOBs

Use the following options to configure a column to store MOBs:

- `IS_MOB` is a Boolean option, which specifies whether or not the column can store MOBs.
- `MOB_THRESHOLD` configures the number of bytes at which an object is considered to be a MOB. If you do not specify a value for `MOB_THRESHOLD`, the default is 100 KB. If you write a value larger than this threshold, it is treated as a MOB.

You can configure a column to store MOBs using the HBase Shell or the Java API.

**Using HBase Shell:**

```
hbase> create 't1', {NAME => 'f1', IS_MOB => true, MOB_THRESHOLD => 102400}
hbase> alter 't1', {NAME => 'f1', IS_MOB => true, MOB_THRESHOLD =>
102400}
```

**Using the Java API:**

```
HColumnDescriptor hcd = new HColumnDescriptor("f");
hcd.setMobEnabled(true);
hcd.setMobThreshold(102400L);
```

## HBase MOB Cache Properties

Because there can be a large number of MOB files at any time, as compared to the number of HFiles, MOB files are not always kept open. The MOB file reader cache is a LRU cache which keeps the most recently used MOB files open.

The following properties are available for tuning the HBase MOB cache.

**Table 13: HBase MOB Cache Properties**

| Property | Default | Description |
|---|---|---|
| `hbase.mob.file.cache.size` | 1000 | The of opened file handlers to cache. A larger value will benefit reads by providing more file handlers per MOB file cache and would reduce frequent file opening and closing of files. However, if the value is too high, errors such as "Too many opened file handlers" may be logged. |
| `hbase.mob.cache.evict.period` | 3600 | The amount of time in seconds after a file is opened before the MOB cache |

| Property | Default | Description |
|---|---|---|
| | | evicts cached files. The default value is 3600 seconds. |
| hbase.mob.cache.evict.remain.ratio | 0.5f | The ratio, expressed as a float between 0.0 and 1.0, that controls how manyfiles remain cached after an eviction is triggered due to the number of cached files exceeding the hbase.mob.file.cache.size. The default value is 0.5f. |

## Configuring the MOB Cache Using Cloudera Manager

To configure the MOB cache within Cloudera Manager, edit the HBase Service advanced configuration snippet for the cluster. Cloudera recommends testing your configuration with the default settings first.

1. Go to the HBase service.
2. Click the **Configuration** tab.
3. Search for the property **HBase Service Advanced Configuration Snippet (Safety Valve)** for `hbase-site.xml`.
4. Paste your configuration into the **Value** field and save your changes. The following example sets the `hbase.mob.cache.evict.period` property to 5000 seconds. See Table 13: HBase MOB Cache Properties on page 46 for a full list of configurable properties for HBase MOB.

```
<property>
  <name>hbase.mob.cache.evict.period</name>
  <value>5000</value>
</property>
```

5. Restart your cluster for the changes to take effect.

## Testing MOB Storage and Retrieval Performance

HBase provides the Java utility `org.apache.hadoop.hbase.IntegrationTestIngestMOB` to assist with testing the MOB feature and deciding on appropriate configuration values for your situation. The utility is run as follows:

```
$ sudo -u hbase hbase org.apache.hadoop.hbase.IntegrationTestIngestMOB \
            -threshold 102400 \
            -minMobDataSize 512 \
            -maxMobDataSize 5120
```

- **`threshold`** is the threshold at which cells are considered to be MOBs. The default is 1 kB, expressed in bytes.
- **`minMobDataSize`** is the minimum value for the size of MOB data. The default is 512 B, expressed in bytes.
- **`maxMobDataSize`** is the maximum value for the size of MOB data. The default is 5 kB, expressed in bytes.

## Compacting MOB Files Manually

You can trigger manual compaction of MOB files manually, rather than waiting for them to be triggered by your configuration, using the `compact` and `major_compact` HBase Shell commands. For MOB, each of these commands requires these parameters in the following order:

1. The table name.
2. An optional column family name or `nil`.
3. The keyword `MOB`.

If the column family is provided, only that column family's files are compacted. Otherwise, all MOB-enabled column families' files are compacted.

```
hbase> compact 't1', nil, 'MOB'
hbase> compact 't1', 'c1', 'MOB'
hbase> major_compact 't1', nil, 'MOB'
hbase> major_compact 't1', 'c1', 'MOB'
```

This functionality is also available using the API, using the `Admin.compact` and `Admin.majorCompact` methods.

## Configuring the Storage Policy for the Write-Ahead Log (WAL)

In CDH 5.7.0 and higher, you can configure the preferred HDFS storage policy for HBase's write-ahead log (WAL) replicas. This feature allows you to tune HBase's use of SSDs to your available resources and the demands of your workload.

These instructions assume that you have followed the instructions in Configuring Storage Directories for DataNodes and that your cluster has SSD storage available to HBase. If HDFS is not configured to use SSDs, these configuration changes will have no effect on HBase. The following policies are available:

- **NONE**: no preference about where the replicas are written.
- **ONE_SSD**: place one replica on SSD storage and the remaining replicas in default storage. This allows you to derive some benefit from SSD storage even if it is a scarce resource in your cluster.

  **Warning: `ONE_SSD`** mode has not been thoroughly tested with HBase and is not recommended.

- **ALL_SSD**: place all replicas on SSD storage.

### Configuring the Storage Policy for WALs Using Cloudera Manager

**Minimum Required Role: Full Administrator**

1. Go to the HBase service.
2. Click the **Configuration** tab.
3. Search for the property **WAL HSM Storage Policy**.
4. Select your desired storage policy.
5. Save your changes. Restart all HBase roles.

Changes will take effect after the next major compaction.

### Configuring the Storage Policy for WALs Using the Command Line

**Important:**

- Follow these command-line instructions on systems that do not use Cloudera Manager.
- This information applies specifically to CDH 6.3.x. See Cloudera Documentation for information specific to other releases.

Paste the following XML into `hbase-site.xml`. Uncomment the <value> line that corresponds to your desired storage policy.

```
<property>
  <name>hbase.wal.storage.policy</name>
  <value>NONE</value>
  <!--<value>ONE_SSD</value>-->
  <!--<value>ALL_SSD</value>-->
</property>
```

**Warning:** **`ONE_SSD`** mode has not been thoroughly tested with HBase and is not recommended.

Restart HBase. Changes will take effect for a given region during its next major compaction.

# Managing HBase

Cloudera Manager requires certain additional steps to set up and configure the HBase service.

## Creating the HBase Root Directory

**Minimum Required Role:** Cluster Administrator (also provided by **Full Administrator**)

When adding the HBase service, the **Add Service** wizard automatically creates a root directory for HBase in HDFS. If you quit the **Add Service** wizard or it does not finish, you can create the root directory outside the wizard by doing these steps:

1. Choose **Create Root Directory** from the **Actions** menu in the **HBase > Status** tab.
2. Click **Create Root Directory** again to confirm.

## Graceful Shutdown

**Minimum Required Role:** Operator (also provided by **Configurator**, **Cluster Administrator**, **Full Administrator**)

A graceful shutdown of an HBase RegionServer allows the regions hosted by that RegionServer to be moved to other RegionServers before stopping the RegionServer. Cloudera Manager provides the following configuration options to perform a graceful shutdown of either an HBase RegionServer or the entire service.

To increase the speed of a rolling restart of the HBase service, set the **Region Mover Threads** property to a higher value. This increases the number of regions that can be moved in parallel, but places additional strain on the HMaster. In most cases, **Region Mover Threads** should be set to 5 or lower.

### Gracefully Shutting Down an HBase RegionServer

1. Go to the HBase service.
2. Click the **Instances** tab.
3. From the list of Role Instances, select the RegionServer you want to shut down gracefully.
4. Select **Actions for Selected** > **Decommission (Graceful Stop)**.
5. Cloudera Manager attempts to gracefully shut down the RegionServer for the interval configured in the **Graceful Shutdown Timeout** configuration option, which defaults to 3 minutes. If the graceful shutdown fails, Cloudera Manager forcibly stops the process by sending a `SIGKILL` (`kill -9`) signal. HBase will perform recovery actions on regions that were on the forcibly stopped RegionServer.
6. If you cancel the graceful shutdown before the **Graceful Shutdown Timeout** expires, you can still manually stop a RegionServer by selecting **Actions for Selected** > **Stop**, which sends a `SIGTERM` (`kill -5`) signal.

### Gracefully Shutting Down the HBase Service

1. Go to the HBase service.
2. Select **Actions** > **Stop**. This tries to perform an HBase Master-driven graceful shutdown for the length of the configured Graceful Shutdown Timeout (three minutes by default), after which it abruptly shuts down the whole service.

### Configuring the Graceful Shutdown Timeout Property

**Minimum Required Role:** Configurator (also provided by **Cluster Administrator, Full Administrator**)

This timeout only affects a graceful shutdown of the entire HBase service, not individual RegionServers. Therefore, if you have a large cluster with many RegionServers, you should strongly consider increasing the timeout from its default of 180 seconds.

1. Go to the HBase service.
2. Click the **Configuration** tab.
3. Select **Scope** > **HBASE-1 (Service Wide)**
4. Use the Search box to search for the **Graceful Shutdown Timeout** property and edit the value.
5. Click **Save Changes** to save this setting.

## Configuring the HBase Thrift Server Role

**Minimum Required Role:** Cluster Administrator (also provided by **Full Administrator**)

The Thrift Server role is not added by default when you install HBase, but it is required before you can use certain other features such as the Hue HBase browser. To add the Thrift Server role:

1. Go to the HBase service.
2. Click the **Instances** tab.
3. Click the **Add Role Instances** button.
4. Select the host(s) where you want to add the Thrift Server role (you only need one for Hue) and click **Continue**. The Thrift Server role should appear in the instances list for the HBase server.
5. Select the Thrift Server role instance.
6. Select **Actions for Selected** > **Start**.

## Enabling HBase Indexing

**Minimum Required Role:** Configurator (also provided by **Cluster Administrator, Full Administrator**)

HBase indexing is dependent on the Key-Value Store Indexer service. The Key-Value Store Indexer service uses the Lily HBase Indexer Service to index the stream of records being added to HBase tables. Indexing allows you to query data stored in HBase with the Solr service.

1. Go to the HBase service.
2. Click the **Configuration** tab.
3. Select **Scope** > **HBASE-1 (Service Wide)**
4. Select **Category** > **Backup**.
5. Select the **Enable Replication** and **Enable Indexing** properties.
6. Click **Save Changes**.

## Adding a Custom Coprocessor

**Minimum Required Role:** Configurator (also provided by **Cluster Administrator, Full Administrator**)

The HBase coprocessor framework provides a way to extend HBase with custom functionality. To configure these properties in Cloudera Manager:

1. Select the HBase service.
2. Click the **Configuration** tab.
3. Select **Scope** > **All**.
4. Select **Category** > **All**.
5. Type `HBase Coprocessor` in the Search box.
6. You can configure the values of the following properties:
   - **HBase Coprocessor Abort on Error** (Service-Wide)
   - **HBase Coprocessor Master Classes** (Master Default Group)
   - **HBase Coprocessor Region Classes** (RegionServer Default Group)
7. Enter a **Reason for change**, and then click **Save Changes** to commit the changes.

## Disabling Loading of Coprocessors

**Minimum Required Role:** Configurator (also provided by **Cluster Administrator, Full Administrator**)

In CDH 5.7 and higher, you can disable loading of system (HBase-wide) or user (table-wide) coprocessors. Cloudera recommends against disabling loading of system coprocessors, because HBase security functionality is implemented using system coprocessors. However, disabling loading of user coprocessors may be appropriate.

1. Select the HBase service.
2. Click the **Configuration** tab.
3. Search for **HBase Service Advanced Configuration Snippet (Safety Valve) for hbase-site.xml**.
4. To disable loading of all coprocessors, add a new property with the name `hbase.coprocessor.enabled` and set its value to `false`. **Cloudera does not recommend this setting.**
5. To disable loading of user coprocessors, add a new property with the name `hbase.coprocessor.user.enabled` and set its value to `false`.
6. Enter a **Reason for change**, and then click **Save Changes** to commit the changes.

## Enabling Hedged Reads on HBase

**Minimum Required Role:** Configurator (also provided by **Cluster Administrator, Full Administrator**)

1. Go to the HBase service.
2. Click the **Configuration** tab.
3. Select **Scope** > **HBASE-1 (Service-Wide)**.
4. Select **Category** > **Performance**.
5. Configure the **HDFS Hedged Read Threadpool Size** and **HDFS Hedged Read Delay Threshold** properties. The descriptions for each of these properties on the configuration pages provide more information.
6. Enter a **Reason for change**, and then click **Save Changes** to commit the changes.

## Moving HBase Master Role to Another Host

**Minimum Required Role:** Cluster Administrator (also provided by **Full Administrator**)

1. Select the HBase service in Cloudera Manager.
2. Click the **Instances** tab.
3. Click **Add Role Instances** to add role instances to HBase.
4. Choose the desired new host under **Master**.
5. Click **Continue**.
6. Start the newly added HBase Master role.

   As a results the state of this role becomes `Started`.

7. Wait until the type of the newly added HBase Master role becomes `Master (Backup)`.
8. Stop any other non-active HBase Master role instances.

   **Note:** This step does not impact HBase. It is required to ensure that the newly created HBase Master role backup will be chosen to be the new active HBase Master role.

9. Stop the remaining active HBase Master role.

   As a result, the type of the newly added HBase Master role automatically becomes `Master (Active)`.

10. Delete the old HBase Master role instances on hosts that are not wanted.

## Advanced Configuration for Write-Heavy Workloads

HBase includes several advanced configuration parameters for adjusting the number of threads available to service flushes and compactions in the presence of write-heavy workloads. Tuning these parameters incorrectly can severely degrade performance and is not necessary for most HBase clusters. If you use Cloudera Manager, configure these options using the **HBase Service Advanced Configuration Snippet (Safety Valve) for hbase-site.xml**.

**`hbase.hstore.flusher.count`**

The number of threads available to flush writes from memory to disk. Never increase `hbase.hstore.flusher.count` to more of 50% of the number of disks available to HBase. For example, if you have 8 solid-state drives (SSDs), `hbase.hstore.flusher.count` should never exceed 4. This allows scanners and compactions to proceed even in the presence of very high writes.

**`hbase.regionserver.thread.compaction.large` and `hbase.regionserver.thread.compaction.small`**

The number of threads available to handle small and large compactions, respectively. Never increase either of these options to more than 50% of the number of disks available to HBase.

Ideally, `hbase.regionserver.thread.compaction.small` should be greater than or equal to `hbase.regionserver.thread.compaction.large`, since the large compaction threads do more intense work and will be in use longer for a given operation.

In addition to the above, if you use compression on some column families, more CPU will be used when flushing these column families to disk during flushes or compaction. The impact on CPU usage depends on the size of the flush or the amount of data to be decompressed and compressed during compactions.

## Starting and Stopping HBase

Use these instructions to start, stop, restart, rolling restart, or decommission HBase clusters or individual hosts.

### Starting or Restarting HBase

You can start HBase hosts individually or as an entire cluster.

1. Go to the HBase service.
2. Click the **Actions** button and select **Start**.
3. To restart a running cluster, click **Actions** and select **Restart** or **Rolling Restart**. A rolling restart, which restarts each RegionServer, one at a time, after a grace period. To configure the grace period, see Configuring the Graceful Shutdown Timeout Property on page 50.
4. The Thrift service has no dependencies and can be restarted at any time. To stop or restart the Thrift service:
   - Go to the HBase service.
   - Select Instances.
   - Select the **HBase Thrift Server** instance.
   - Select **Actions for Selected** and select either **Stop** or **Restart**.

### Stopping HBase

You can stop a single HBase host, all hosts of a given type, or all hosts in the cluster.

1. To stop or decommission a single RegionServer:
   a. Go to the HBase service.
   b. Click the **Instances** tab.
   c. From the list of Role Instances, select the RegionServer or RegionServers you want to stop or decommission.
   d. Select **Actions for Selected** and select either **Decommission (Graceful Stop)** or **Stop**.
      - **Graceful Stop** causes the regions to be redistributed to other RegionServers, increasing availability during the RegionServer outage. Cloudera Manager waits for an interval determined by the Graceful Shutdown

timeout interval, which defaults to three minutes. If the graceful stop does not succeed within this interval, the RegionServer is stopped with a `SIGKILL` (`kill -9`) signal. Recovery will be initiated on affected regions.

- **Stop** happens immediately and does not redistribute the regions. It issues a `SIGTERM` (`kill -5`) signal.

2. To stop or decommission a single HMaster, select the Master and go through the same steps as above.
3. To stop or decommission the entire cluster, select the **Actions** button at the top of the screen (not **Actions for selected**) and select **Decommission (Graceful Stop)** or **Stop**.

## Accessing HBase by using the HBase Shell

After you have started HBase, you can access the database in an interactive way by using the HBase Shell, which is a command interpreter for HBase which is written in Ruby. Always run HBase administrative commands such as the HBase Shell, `hbck`, or bulk-load commands as the HBase user (typically `hbase`).

```
hbase shell
```

### HBase Shell Overview

- To get help and to see all available commands, use the `help` command.
- To get help on a specific command, use `help "command"`. For example:

```
hbase> help "create"
```

- To remove an attribute from a table or column family or reset it to its default value, set its value to `nil`. For example, use the following command to remove the `KEEP_DELETED_CELLS` attribute from the `f1` column of the `users` table:

```
hbase> alter 'users', { NAME => 'f1', KEEP_DELETED_CELLS => nil }
```

- To exit the HBase Shell, type `quit`.

### Setting Virtual Machine Options for HBase Shell

HBase in CDH 5.2 and higher allows you to set variables for the virtual machine running HBase Shell, by using the `HBASE_SHELL_OPTS` environment variable. This example sets several options in the virtual machine.

```
$ HBASE_SHELL_OPTS="-verbose:gc -XX:+PrintGCApplicationStoppedTime -XX:+PrintGCDateStamps
      -XX:+PrintGCDetails -Xloggc:$HBASE_HOME/logs/gc-hbase.log" ./bin/hbase shell
```

### Scripting with HBase Shell

CDH 5.2 and higher include non-interactive mode. This mode allows you to use HBase Shell in scripts, and allow the script to access the exit status of the HBase Shell commands. To invoke non-interactive mode, use the `-n` or `--non-interactive` switch. This small example script shows how to use HBase Shell in a Bash script.

```
#!/bin/bash
echo 'list' | hbase shell -n
status=$?
if [ $status -ne 0 ]; then
  echo "The command may have failed."
fi
```

Successful HBase Shell commands return an exit status of 0. However, an exit status other than 0 does not necessarily indicate a failure, but should be interpreted as unknown. For example, a command may succeed, but while waiting for the response, the client may lose connectivity. In that case, the client has no way to know the outcome of the command.

In the case of a non-zero exit status, your script should check to be sure the command actually failed before taking further action.

CDH 5.7 and higher include the `get_splits` command, which returns the split points for a given table:

```
hbase> get_splits 't2'
Total number of splits = 5

=> ["", "10", "20", "30", "40"]
```

You can also write Ruby scripts for use with HBase Shell. Example Ruby scripts are included in the `hbase-examples/src/main/ruby/` directory.

## Using HBase Command-Line Utilities

Besides the HBase Shell, HBase includes several other command-line utilities, which are available in the `hbase/bin/` directory of each HBase host. This topic provides basic usage instructions for the most commonly used utilities.

### `PerformanceEvaluation`

The `PerformanceEvaluation` utility allows you to run several preconfigured tests on your cluster and reports its performance. To run the `PerformanceEvaluation` tool in CDH 5.1 and higher, use the `bin/hbase pe` command. In CDH 5.0 and lower, use the command `bin/hbase org.apache.hadoop.hbase.PerformanceEvaluation`.

For usage instructions, run the command with no arguments. The following output shows the usage instructions for the `PerformanceEvaluation` tool in CDH 5.7. Options and commands available depend on the CDH version.

```
$ hbase pe

Usage: java org.apache.hadoop.hbase.PerformanceEvaluation \
  <OPTIONS> [-D<property=value>]* <command> <nclients>

Options:
 nomapred        Run multiple clients using threads (rather than use mapreduce)
 rows            Rows each client runs. Default: One million
 size            Total size in GiB. Mutually exclusive with --rows. Default: 1.0.
 sampleRate      Execute test on a sample of total rows. Only supported by randomRead.
                 Default: 1.0
 traceRate       Enable HTrace spans. Initiate tracing every N rows. Default: 0
 table           Alternate table name. Default: 'TestTable'
 multiGet        If >0, when doing RandomRead, perform multiple gets instead of single
                 gets.
                 Default: 0
 compress        Compression type to use (GZ, LZO, ...). Default: 'NONE'
 flushCommits    Used to determine if the test should flush the table. Default: false
 writeToWAL      Set writeToWAL on puts. Default: True
 autoFlush       Set autoFlush on htable. Default: False
 oneCon          all the threads share the same connection. Default: False
 presplit        Create presplit table. Recommended for accurate perf analysis (see
                 guide). Default: disabled
 inmemory        Tries to keep the HFiles of the CF inmemory as far as possible. Not
                 guaranteed that reads are always served from memory.  Default: false
 usetags         Writes tags along with KVs. Use with HFile V3. Default: false
 numoftags       Specify the no of tags that would be needed. This works only if usetags
                 is true.
 filterAll       Helps to filter out all the rows on the server side there by not returning
                 anything back to the client.  Helps to check the server side performance.
                 Uses FilterAllFilter internally.
 latency         Set to report operation latencies. Default: False
 bloomFilter     Bloom filter type, one of [NONE, ROW, ROWCOL]
 valueSize       Pass value size to use: Default: 1024
 valueRandom     Set if we should vary value size between 0 and 'valueSize'; set on read
                 for stats on size: Default: Not set.
 valueZipf       Set if we should vary value size between 0 and 'valueSize' in zipf form:
                 Default: Not set.
 period          Report every 'period' rows: Default: opts.perClientRunRows / 10
 multiGet        Batch gets together into groups of N. Only supported by randomRead.
                 Default: disabled
 addColumns      Adds columns to scans/gets explicitly. Default: true
 replicas        Enable region replica testing. Defaults: 1.
 splitPolicy     Specify a custom RegionSplitPolicy for the table.
 randomSleep     Do a random sleep before each get between 0 and entered value. Defaults: 0
```

```
 columns         Columns to write per row. Default: 1
 caching         Scan caching to use. Default: 30

 Note: -D properties will be applied to the conf used.
  For example:
   -Dmapreduce.output.fileoutputformat.compress=true
   -Dmapreduce.task.timeout=60000

Command:
 append          Append on each row; clients overlap on keyspace so some concurrent
                 operations
 checkAndDelete  CheckAndDelete on each row; clients overlap on keyspace so some concurrent
                 operations
 checkAndMutate  CheckAndMutate on each row; clients overlap on keyspace so some concurrent
                 operations
 checkAndPut     CheckAndPut on each row; clients overlap on keyspace so some concurrent
                 operations
 filterScan      Run scan test using a filter to find a specific row based on it's value
                 (make sure to use --rows=20)
 increment       Increment on each row; clients overlap on keyspace so some concurrent
                 operations
 randomRead      Run random read test
 randomSeekScan  Run random seek and scan 100 test
 randomWrite     Run random write test
 scan            Run scan test (read every row)
 scanRange10     Run random seek scan with both start and stop row (max 10 rows)
 scanRange100    Run random seek scan with both start and stop row (max 100 rows)
 scanRange1000   Run random seek scan with both start and stop row (max 1000 rows)
 scanRange10000  Run random seek scan with both start and stop row (max 10000 rows)
 sequentialRead  Run sequential read test
 sequentialWrite Run sequential write test

Args:
 nclients        Integer. Required. Total number of clients (and HRegionServers)
                 running: 1 <= value <= 500
Examples:
 To run a single client doing the default 1M sequentialWrites:
 $ bin/hbase org.apache.hadoop.hbase.PerformanceEvaluation sequentialWrite 1
 To run 10 clients doing increments over ten rows:
 $ bin/hbase org.apache.hadoop.hbase.PerformanceEvaluation --rows=10 --nomapred increment 10
```

### LoadTestTool

The `LoadTestTool` utility load-tests your cluster by performing writes, updates, or reads on it. To run the `LoadTestTool` in CDH 5.1 and higher, use the `bin/hbase ltt` command . In CDH 5.0 and lower, use the command `bin/hbase org.apache.hadoop.hbase.util.LoadTestTool`. To print general usage information, use the -h option. Options and commands available depend on the CDH version.

```
$ bin/hbase ltt -h

Options:
 -batchupdate                   Whether to use batch as opposed to separate updates for every
column
                                in a row
 -bloom <arg>                   Bloom filter type, one of [NONE, ROW, ROWCOL]
 -compression <arg>             Compression type, one of [LZO, GZ, NONE, SNAPPY, LZ4]
 -data_block_encoding <arg>     Encoding algorithm (e.g. prefix compression) to use for data
blocks
                                in the test column family, one of
                                [NONE, PREFIX, DIFF, FAST_DIFF, PREFIX_TREE].
 -deferredlogflush              Enable deferred log flush.
 -encryption <arg>              Enables transparent encryption on the test table, one of [AES]
 -families <arg>                The name of the column families to use separated by comma
 -generator <arg>               The class which generates load for the tool. Any args for this
class
                                can be passed as colon separated after class name
 -h,--help                      Show usage
 -in_memory                     Tries to keep the HFiles of the CF inmemory as far as possible.
 Not
                                guaranteed that reads are always served from inmemory
 -init_only                     Initialize the test table only, don't do any loading
 -key_window <arg>             The 'key window' to maintain between reads and writes for concurrent

                                write/read workload. The default is 0.
 -max_read_errors <arg>         The maximum number of read errors to tolerate before terminating
 all
                                reader threads. The default is 10.
 -mob_threshold <arg>           Desired cell size to exceed in bytes that will use the MOB write
 path
```

```
 -multiget_batchsize <arg>       Whether to use multi-gets as opposed to separate gets for every
                                 column in a row
 -multiput                       Whether to use multi-puts as opposed to separate puts for every
                                 column in a row
 -num_keys <arg>                 The number of keys to read/write
 -num_regions_per_server <arg>   Desired number of regions per region server. Defaults to 5.
 -num_tables <arg>               A positive integer number. When a number n is specified, load
test tool
                                 will load n table parallely. -tn parameter value becomes table
name prefix.
                                 Each table name is in format <tn>_1...<tn>_n
 -read <arg>                     <verify_percent>[:<#threads=20>]
 -reader <arg>                   The class for executing the read requests
 -region_replica_id <arg>        Region replica id to do the reads from
 -region_replication <arg>       Desired number of replicas per region
 -regions_per_server <arg>       A positive integer number. When a number n is specified, load
test tool
                                 will create the test table with n regions per server
 -skip_init                      Skip the initialization; assume test table already exists
 -start_key <arg>                The first key to read/write (a 0-based index). The default value
 is 0.
 -tn <arg>                       The name of the table to read or write
 -update <arg>                   <update_percent>[:<#threads=20>][:<#whether to ignore nonce
collisions=0>]
 -updater <arg>                  The class for executing the update requests
 -write <arg>                    <avg_cols_per_key>:<avg_data_size>[:<#threads=20>]
 -writer <arg>                   The class for executing the write requests
 -zk <arg>                       ZK quorum as comma-separated host names without port numbers
 -zk_root <arg>                  name of parent znode in zookeeper
```

### `wal`

The `wal` utility prints information about the contents of a specified WAL file. To get a list of all WAL files, use the HDFS command `hadoop fs -ls -R /hbase/WALs`. To run the `wal` utility, use the `bin/hbase wal` command. Run it without options to get usage information.

```
hbase wal
usage: WAL <filename...> [-h] [-j] [-p] [-r <arg>] [-s <arg>] [-w <arg>]
 -h,--help             Output help message
 -j,--json             Output JSON
 -p,--printvals        Print values
 -r,--region <arg>     Region to filter by. Pass encoded region name; e.g.
                       '9192caead6a5a20acb4454ffbc79fa14'
 -s,--sequence <arg>   Sequence to filter by. Pass sequence number.
 -w,--row <arg>        Row to filter by. Pass row name.
```

### `hfile`

The `hfile` utility prints diagnostic information about a specified `hfile`, such as block headers or statistics. To get a list of all `hfiles`, use the HDFS command `hadoop fs -ls -R /hbase/data`. To run the `hfile` utility, use the `bin/hbase hfile` command. Run it without options to get usage information.

```
$ hbase hfile

usage: HFile [-a] [-b] [-e] [-f <arg> | -r <arg>] [-h] [-i] [-k] [-m] [-p]
       [-s] [-v] [-w <arg>]
 -a,--checkfamily         Enable family check
 -b,--printblocks         Print block index meta data
 -e,--printkey            Print keys
 -f,--file <arg>          File to scan. Pass full-path; e.g.
                          hdfs://a:9000/hbase/hbase:meta/12/34
 -h,--printblockheaders   Print block headers for each block.
 -i,--checkMobIntegrity   Print all cells whose mob files are missing
 -k,--checkrow            Enable row order check; looks for out-of-order
                          keys
 -m,--printmeta           Print meta data of file
 -p,--printkv             Print key/value pairs
 -r,--region <arg>        Region to scan. Pass region name; e.g.
                          'hbase:meta,,1'
 -s,--stats               Print statistics
 -v,--verbose             Verbose output; emits file and meta data
                          delimiters
 -w,--seekToRow <arg>     Seek to this row and print all the kvs for this
                          row only
```

### hbck

The hbck utility checks and optionally repairs errors in HFiles.

> **Warning:** Running hbck with any of the `-fix` or `-repair` commands is dangerous and can lead to data loss. Contact Cloudera support before running it.

To run hbck, use the `bin/hbase hbck` command. Run it with the `-h` option to get more usage information.

```
-------------------------------------------------------------------------
NOTE: As of HBase version 2.0, the hbck tool is significantly changed.
In general, all Read-Only options are supported and can be be used
safely. Most -fix/ -repair options are NOT supported. Please see usage
below for details on which options are not supported.
-------------------------------------------------------------------------

Usage: fsck [opts] {only tables}
 where [opts] are:
   -help Display help options (this)
   -details Display full report of all regions.
   -timelag <timeInSeconds>  Process only regions that  have not experienced any metadata updates
 in the last  <timeInSeconds> seconds.
   -sleepBeforeRerun <timeInSeconds> Sleep this many seconds before checking if the fix worked if
 run with -fix
   -summary Print only summary of the tables and status.
   -metaonly Only check the state of the hbase:meta table.
   -sidelineDir <hdfs://> HDFS path to backup existing meta.
   -boundaries Verify that regions boundaries are the same between META and store files.
   -exclusive Abort if another hbck is exclusive or fixing.

  Datafile Repair options: (expert features, use with caution!)
   -checkCorruptHFiles     Check all Hfiles by opening them to make sure they are valid
   -sidelineCorruptHFiles  Quarantine corrupted HFiles.  implies -checkCorruptHFiles

 Replication options
   -fixReplication   Deletes replication queues for removed peers

  Metadata Repair options supported as of version 2.0: (expert features, use with caution!)
   -fixVersionFile   Try to fix missing hbase.version file in hdfs.
   -fixReferenceFiles  Try to offline lingering reference store files
   -fixHFileLinks  Try to offline lingering HFileLinks
   -noHdfsChecking   Don't load/check region info from HDFS. Assumes hbase:meta region info is
good. Won't check/fix any HDFS issue, e.g. hole, orphan, or overlap
   -ignorePreCheckPermission  ignore filesystem permission pre-check

NOTE: Following options are NOT supported as of HBase version 2.0+.

  UNSUPPORTED Metadata Repair options: (expert features, use with caution!)
   -fix              Try to fix region assignments.  This is for backwards compatiblity
   -fixAssignments   Try to fix region assignments.  Replaces the old -fix
   -fixMeta          Try to fix meta problems.  This assumes HDFS region info is good.
   -fixHdfsHoles     Try to fix region holes in hdfs.
   -fixHdfsOrphans   Try to fix region dirs with no .regioninfo file in hdfs
   -fixTableOrphans  Try to fix table dirs with no .tableinfo file in hdfs (online mode only)
   -fixHdfsOverlaps  Try to fix region overlaps in hdfs.
   -maxMerge <n>     When fixing region overlaps, allow at most <n> regions to merge. (n=5 by
default)
   -sidelineBigOverlaps  When fixing region overlaps, allow to sideline big overlaps
   -maxOverlapsToSideline <n>  When fixing region overlaps, allow at most <n> regions to sideline
 per group. (n=2 by default)
   -fixSplitParents  Try to force offline split parents to be online.
   -removeParents    Try to offline and sideline lingering parents and keep daughter regions.
   -fixEmptyMetaCells  Try to fix hbase:meta entries not referencing any region (empty
REGIONINFO_QUALIFIER rows)

  UNSUPPORTED Metadata Repair shortcuts
   -repair           Shortcut for -fixAssignments -fixMeta -fixHdfsHoles -fixHdfsOrphans
-fixHdfsOverlaps -fixVersionFile -sidelineBigOverlaps -fixReferenceFiles-fixHFileLinks
   -repairHoles      Shortcut for -fixAssignments -fixMeta -fixHdfsHoles
```

### clean

After you have finished using a test or proof-of-concept cluster, the `hbase clean` utility can remove all HBase-related data from ZooKeeper and HDFS.

**Warning:** The `hbase clean` command destroys data. Do not run it on production clusters, or unless you are absolutely sure you want to destroy the data.

To run the `hbase clean` utility, use the `bin/hbase clean` command. Run it with no options for usage information.

```
$ bin/hbase clean

Usage: hbase clean (--cleanZk|--cleanHdfs|--cleanAll)
Options:
        --cleanZk   cleans hbase related data from zookeeper.
        --cleanHdfs cleans hbase related data from hdfs.
        --cleanAll  cleans hbase related data from both zookeeper and hdfs.
```

## Writing Data to HBase

To write data to HBase, you use methods of the `Table` class. You can use the Java API directly, or use the HBase Shell, the REST API, the Thrift API, , or another client which uses the Java API indirectly. When you issue a Put, the coordinates of the data are the row, the column, and the timestamp. The timestamp is unique per version of the cell, and can be generated automatically or specified programmatically by your application, and must be a long integer.

### Variations on Put

There are several different ways to write data into HBase. Some of them are listed below.

- A `Put` operation writes data into HBase.
- A `Delete` operation deletes data from HBase. What actually happens during a Delete depends upon several factors.
- A `CheckAndPut` operation performs a Scan before attempting the `Put`, and only does the `Put` if a value matches what is expected, and provides row-level atomicity.
- A `CheckAndDelete` operation performs a Scan before attempting the `Delete`, and only does the `Delete` if a value matches what is expected.
- An `Increment` operation increments values of one or more columns within a single row, and provides row-level atomicity.

Refer to the API documentation for a full list of methods provided for writing data to HBase.Different methods require different access levels and have other differences.

### Versions

When you put data into HBase, a timestamp is required. The timestamp can be generated automatically by the RegionServer or can be supplied by you. The timestamp must be unique per version of a given cell, because the timestamp identifies the version. To modify a previous version of a cell, for instance, you would issue a Put with a different value for the data itself, but the same timestamp.

HBase's behavior regarding versions is highly configurable. The maximum number of versions defaults to 1. You can change the default value for HBase by configuring `hbase.column.max.version` in `hbase-site.xml`, either using an advanced configuration snippet if you use Cloudera Manager, or by editing the file directly otherwise.

You can also configure the maximum and minimum number of versions to keep for a given column, or specify a default time-to-live (TTL), which is the number of seconds before a version is deleted. The following examples all use `alter` statements in HBase Shell to create new column families with the given characteristics, but you can use the same syntax when creating a new table or to alter an existing column family. This is only a fraction of the options you can specify for a given column family.

```
hbase> alter ‘t1 , NAME => ‘f1 , VERSIONS => 5
hbase> alter ‘t1 , NAME => ‘f1 , MIN_VERSIONS => 2
hbase> alter ‘t1 , NAME => ‘f1 , TTL => 15
```

HBase sorts the versions of a cell from newest to oldest, by sorting the timestamps lexicographically. When a version needs to be deleted because a threshold has been reached, HBase always chooses the "oldest" version, even if it is in fact the most recent version to be inserted. Keep this in mind when designing your timestamps. Consider using the default generated timestamps and storing other version-specific data elsewhere in the row, such as in the row key. If `MIN_VERSIONS` and `TTL` conflict, `MIN_VERSIONS` takes precedence.

### Deletion

When you request for HBase to delete data, either explicitly using a Delete method or implicitly using a threshold such as the maximum number of versions or the TTL, HBase does not delete the data immediately. Instead, it writes a deletion marker, called a tombstone, to the HFile, which is the physical file where a given RegionServer stores its region of a column family. The tombstone markers are processed during major compaction operations, when HFiles are rewritten without the deleted data included.

Even after major compactions, "deleted" data may not actually be deleted. You can specify the `KEEP_DELETED_CELLS` option for a given column family, and the tombstones will be preserved in the HFile even after major compaction. One scenario where this approach might be useful is for data retention policies.

Another reason deleted data may not actually be deleted is if the data would be required to restore a table from a snapshot which has not been deleted. In this case, the data is moved to an archive during a major compaction, and only deleted when the snapshot is deleted. This is a good reason to monitor the number of snapshots saved in HBase.

### Examples

This abbreviated example writes data to an HBase table using HBase Shell and then scans the table to show the result.

```
hbase> put 'test', 'row1', 'cf:a', 'value1'
0 row(s) in 0.1770 seconds

hbase> put 'test', 'row2', 'cf:b', 'value2'
0 row(s) in 0.0160 seconds

hbase> put 'test', 'row3', 'cf:c', 'value3'
0 row(s) in 0.0260 seconds
hbase> scan 'test'
ROW                   COLUMN+CELL
 row1                 column=cf:a, timestamp=1403759475114, value=value1
 row2                 column=cf:b, timestamp=1403759492807, value=value2
 row3                 column=cf:c, timestamp=1403759503155, value=value3
3 row(s) in 0.0440 seconds
```

This abbreviated example uses the HBase API to write data to an HBase table, using the automatic timestamp created by the Region Server.

```
publicstaticfinalbyte[] CF = "cf".getBytes();
publicstaticfinalbyte[] ATTR = "attr".getBytes();
...
Put put = new Put(Bytes.toBytes(row));
put.add(CF, ATTR, Bytes.toBytes( data));
htable.put(put);
```

This example uses the HBase API to write data to an HBase table, specifying the timestamp.

```
publicstaticfinalbyte[] CF = "cf".getBytes();
publicstaticfinalbyte[] ATTR = "attr".getBytes();
...
Put put = new Put( Bytes.toBytes(row));
long explicitTimeInMs = 555;  // just an example
put.add(CF, ATTR, explicitTimeInMs, Bytes.toBytes(data));
htable.put(put);
```

### Further Reading

- Refer to the Table and HColumnDescriptor API documentation for more details about configuring tables and columns, as well as reading and writing to HBase.
- Refer to the Apache HBase Reference Guide for more in-depth information about HBase, including details about versions and deletions not covered here.

## Importing Data Into HBase

**Note:** This page contains references to CDH 5 components or features that have been removed from CDH 6. These references are only applicable if you are managing a CDH 5 cluster with Cloudera Manager 6. For more information, see Deprecated Items.

The method you use for importing data into HBase depends on several factors:

- The location, size, and format of your existing data
- Whether you need to import data once or periodically over time
- Whether you want to import the data in bulk or stream it into HBase regularly
- How fresh the HBase data needs to be

This topic helps you choose the correct method or composite of methods and provides example workflows for each method.

Always run HBase administrative commands as the HBase user (typically `hbase`).

### Choosing the Right Import Method

**If the data is already in an HBase table:**

- To move the data from one HBase cluster to another, use `snapshot` and either the `clone_snapshot` or `ExportSnapshot` utility; or, use the `CopyTable` utility.
- To move the data from one HBase cluster to another without downtime on either cluster, use replication.

**If the data currently exists outside HBase:**

- If possible, write the data to HFile format, and use a BulkLoad to import it into HBase. The data is immediately available to HBase and you can bypass the normal write path, increasing efficiency.
- If you prefer not to use bulk loads, and you are using a tool such as Pig, you can use it to import your data.

**If you need to stream live data to HBase instead of import in bulk:**

- Write a Java client using the Java API, or use the Apache Thrift Proxy API to write a client in a language supported by Thrift.
- Stream data directly into HBase using the REST Proxy API in conjunction with an HTTP client such as `wget` or `curl`.
- Use Flume or Spark.

Most likely, at least one of these methods works in your situation. If not, you can use MapReduce directly. Test the most feasible methods with a subset of your data to determine which one is optimal.

### Using CopyTable

`CopyTable` uses HBase read and write paths to copy part or all of a table to a new table in either the same cluster or a different cluster. `CopyTable` causes read load when reading from the source, and write load when writing to the destination. Region splits occur on the destination table in real time as needed. To avoid these issues, use `snapshot` and `export` commands instead of `CopyTable`. Alternatively, you can pre-split the destination table to avoid excessive

splits. The destination table can be partitioned differently from the source table. See this section of the Apache HBase documentation for more information.

Edits to the source table after the `CopyTable` starts are not copied, so you may need to do an additional `CopyTable` operation to copy new data into the destination table. Run `CopyTable` as follows, using `--help` to see details about possible parameters.

```
$ ./bin/hbase org.apache.hadoop.hbase.mapreduce.CopyTable --help
Usage: CopyTable [general options] [--starttime=X] [--endtime=Y] [--new.name=NEW]
[--peer.adr=ADR] <tablename>
```

The `starttime`/`endtime` and `startrow`/`endrow` pairs function in a similar way: if you leave out the first of the pair, the first timestamp or row in the table is the starting point. Similarly, if you leave out the second of the pair, the operation continues until the end of the table. To copy the table to a new table in the same cluster, you must specify `--new.name`, unless you want to write the copy back to the same table, which would add a new version of each cell (with the same data), or just overwrite the cell with the same value if the maximum number of versions is set to 1 (the default in CDH 5). To copy the table to a new table in a different cluster, specify `--peer.adr` and optionally, specify a new table name.

The following example creates a new table using HBase Shell in non-interactive mode, and then copies data in two ColumnFamilies in rows starting with timestamp 1265875194289 and including the last row before the CopyTable started, to the new table.

```
echo create 'NewTestTable', 'cf1', 'cf2', 'cf3' | bin/hbase shell --non-interactive
bin/hbase org.apache.hadoop.hbase.mapreduce.CopyTable --starttime=1265875194289
--families=cf1,cf2,cf3 --new.name=NewTestTable TestTable
```

Snapshots are recommended instead of CopyTable for most situations.

## Using Snapshots

Cloudera recommends snapshots instead of CopyTable where possible. A snapshot captures the state of a table at the time the snapshot was taken. Because no data is copied when a snapshot is taken, the process is very quick. As long as the snapshot exists, cells in the snapshot are never deleted from HBase, even if they are explicitly deleted by the API. Instead, they are archived so that the snapshot can restore the table to its state at the time of the snapshot.

After taking a snapshot, use the `clone_snapshot` command to copy the data to a new (immediately enabled) table in the same cluster, or the Export utility to create a new table based on the snapshot, in the same cluster or a new cluster. This is a copy-on-write operation. The new table shares HFiles with the original table until writes occur in the new table but not the old table, or until a compaction or split occurs in either of the tables. This can improve performance in the short term compared to CopyTable.

To export the snapshot to a new cluster, use the `ExportSnapshot` utility, which uses MapReduce to copy the snapshot to the new cluster. Run the `ExportSnapshot` utility on the source cluster, as a user with HBase and HDFS write permission on the destination cluster, and HDFS read permission on the source cluster. This creates the expected amount of IO load on the destination cluster. Optionally, you can limit bandwidth consumption, which affects IO on the destination cluster. After the ExportSnapshot operation completes, you can see the snapshot in the new cluster using the `list_snapshot` command, and you can use the `clone_snapshot` command to create the table in the new cluster from the snapshot.

For full instructions for the `snapshot` and `clone_snapshot` HBase Shell commands, run the HBase Shell and type `help snapshot`. The following example takes a snapshot of a table, uses it to clone the table to a new table in the same cluster, and then uses the `ExportSnapshot` utility to copy the table to a different cluster, with 16 mappers and limited to 200 Mb/sec bandwidth.

```
$ bin/hbase shell
hbase(main):005:0> snapshot 'TestTable', 'TestTableSnapshot'
0 row(s) in 2.3290 seconds

hbase(main):006:0> clone_snapshot 'TestTableSnapshot', 'NewTestTable'
0 row(s) in 1.3270 seconds
```

```
hbase(main):007:0> describe 'NewTestTable'
DESCRIPTION                                          ENABLED
 'NewTestTable', {NAME => 'cf1', DATA_BLOCK_ENCODING true
  => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE
  => '0', VERSIONS => '1', COMPRESSION => 'NONE', MI
 N_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_C
 ELLS => 'false', BLOCKSIZE => '65536', IN_MEMORY =>
  'false', BLOCKCACHE => 'true'}, {NAME => 'cf2', DA
 TA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW',
 REPLICATION_SCOPE => '0', VERSIONS => '1', COMPRESS
 ION => 'NONE', MIN_VERSIONS => '0', TTL => 'FOREVER
 ', KEEP_DELETED_CELLS => 'false', BLOCKSIZE => '655
 36', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
1 row(s) in 0.1280 seconds
hbase(main):008:0> quit

$ hbase org.apache.hadoop.hbase.snapshot.ExportSnapshot -snapshot TestTableSnapshot
-copy-to file:///tmp/hbase -mappers 16 -bandwidth 200
14/10/28 21:48:16 INFO snapshot.ExportSnapshot: Copy Snapshot Manifest
14/10/28 21:48:17 INFO client.RMProxy: Connecting to ResourceManager at
a1221.example.com/192.0.2.121:8032
14/10/28 21:48:19 INFO snapshot.ExportSnapshot: Loading Snapshot 'TestTableSnapshot'
hfile list
14/10/28 21:48:19 INFO Configuration.deprecation: hadoop.native.lib is deprecated.
Instead, use io.native.lib.available
14/10/28 21:48:19 INFO util.FSVisitor: No logs under
directory:hdfs://a1221.example.com:8020/hbase/.hbase-snapshot/TestTableSnapshot/WALs
14/10/28 21:48:20 INFO mapreduce.JobSubmitter: number of splits:0
14/10/28 21:48:20 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_1414556809048_0001
14/10/28 21:48:20 INFO impl.YarnClientImpl: Submitted application
application_1414556809048_0001
14/10/28 21:48:20 INFO mapreduce.Job: The url to track the job:
http://a1221.example.com:8088/proxy/application_1414556809048_0001/
14/10/28 21:48:20 INFO mapreduce.Job: Running job: job_1414556809048_0001
14/10/28 21:48:36 INFO mapreduce.Job: Job job_1414556809048_0001 running in uber mode
: false
14/10/28 21:48:36 INFO mapreduce.Job:  map 0% reduce 0%
14/10/28 21:48:37 INFO mapreduce.Job: Job job_1414556809048_0001 completed successfully
14/10/28 21:48:37 INFO mapreduce.Job: Counters: 2
 Job Counters
  Total time spent by all maps in occupied slots (ms)=0
  Total time spent by all reduces in occupied slots (ms)=0
14/10/28 21:48:37 INFO snapshot.ExportSnapshot: Finalize the Snapshot Export
14/10/28 21:48:37 INFO snapshot.ExportSnapshot: Verify snapshot integrity
14/10/28 21:48:37 INFO Configuration.deprecation: fs.default.name is deprecated. Instead,
 use fs.defaultFS
14/10/28 21:48:37 INFO snapshot.ExportSnapshot: Export Completed: TestTableSnapshot
```

The bold italic line contains the URL from which you can track the `ExportSnapshot` job. When it finishes, a new set of HFiles, comprising all of the HFiles that were part of the table when the snapshot was taken, is created at the HDFS location you specified.

You can use the `SnapshotInfo` command-line utility included with HBase to verify or debug snapshots.

## Using BulkLoad

HBase uses the well-known HFile format to store its data on disk. In many situations, writing HFiles programmatically with your data, and bulk-loading that data into HBase on the RegionServer, has advantages over other data ingest mechanisms. BulkLoad operations bypass the write path completely, providing the following benefits:

- The data is available to HBase immediately but does cause additional load or latency on the cluster when it appears.
- BulkLoad operations do not use the write-ahead log (WAL) and do not cause flushes or split storms.
- BulkLoad operations do not cause excessive garbage collection.

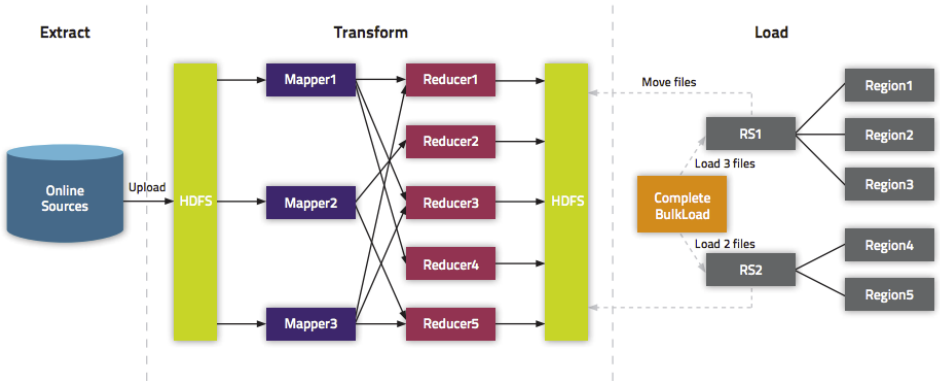
**Note:** Because they bypass the WAL, BulkLoad operations are not propagated between clusters using replication. If you need the data on all replicated clusters, you must perform the BulkLoad on each cluster.

If you use BulkLoads with HBase, your workflow is similar to the following:

1. **Extract your data from its existing source.** For instance, if your data is in a MySQL database, you might run the `mysqldump` command. The process you use depends on your data. If your data is already in TSV or CSV format, skip this step and use the included `ImportTsv` utility to process your data into HFiles. See the ImportTsv documentation for details.
2. **Process your data into HFile format.** See http://hbase.apache.org/book.html#_hfile_format_2 for details about HFile format. Usually you use a MapReduce job for the conversion, and you often need to write the Mapper yourself because your data is unique. The job must to emit the row key as the `Key`, and either a `KeyValue`, a `Put`, or a `Delete` as the `Value`. The Reducer is handled by HBase; configure it using HFileOutputFormat.configureIncrementalLoad() and it does the following:
   - Inspects the table to configure a total order partitioner
   - Uploads the partitions file to the cluster and adds it to the `DistributedCache`
   - Sets the number of `reduce` tasks to match the current number of regions
   - Sets the output key/value class to match `HFileOutputFormat` requirements
   - Sets the Reducer to perform the appropriate sorting (either `KeyValueSortReducer` or `PutSortReducer`)
3. **One HFile is created per region in the output folder.** Input data is almost completely re-written, so you need available disk space at least twice the size of the original data set. For example, for a 100 GB output from `mysqldump`, you should have at least 200 GB of available disk space in HDFS. You can delete the original input file at the end of the process.
4. **Load the files into HBase.** Use the `LoadIncrementalHFiles` command (more commonly known as the completebulkload tool), passing it a URL that locates the files in HDFS. Each file is loaded into the relevant region on the RegionServer for the region. You can limit the number of versions that are loaded by passing the `--versions=` *`N`* option, where *`N`* is the maximum number of versions to include, from newest to oldest (largest timestamp to smallest timestamp).

   If a region was split after the files were created, the tool automatically splits the HFile according to the new boundaries. This process is inefficient, so if your table is being written to by other processes, you should load as soon as the transform step is done.

The following illustration shows the full BulkLoad process.




## Extra Steps for BulkLoad With Encryption Zones

When using BulkLoad to import data into HBase in the a cluster using encryption zones, the following information is important.

- Both the staging directory and the directory into which you place your generated HFiles need to be within HBase's encryption zone (generally under the `/hbase` directory). Before you can do this, you need to change the permissions of `/hbase` to be world-executable but not world-readable (`rwx--x--x`, or numeric mode `711`).

- You also need to configure the HMaster to set the permissions of the HBase root directory correctly. If you use Cloudera Manager, edit the **Master Advanced Configuration Snippet (Safety Valve) for hbase-site.xml**. Otherwise, edit hbase-site.xml on the HMaster. Add the following:

```
<property>
  <name>hbase.rootdir.perms</name>
  <value>711</value>
</property>
```

  If you skip this step, a previously-working BulkLoad setup will start to fail with permission errors when you restart the HMaster.

#### Use Cases for BulkLoad

- **Loading your original dataset into HBase for the first time** - Your initial dataset might be quite large, and bypassing the HBase write path can speed up the process considerably.
- **Incremental Load** - To load new data periodically, use BulkLoad to import it in batches at your preferred intervals. This alleviates latency problems and helps you to achieve service-level agreements (SLAs). However, one trigger for compaction is the number of HFiles on a RegionServer. Therefore, importing a large number of HFiles at frequent intervals can cause major compactions to happen more often than they otherwise would, negatively impacting performance. You can mitigate this by tuning the compaction settings such that the maximum number of HFiles that can be present without triggering a compaction is very high, and relying on other factors, such as the size of the Memstore, to trigger compactions.
- **Data needs to originate elsewhere** - If an existing system is capturing the data you want to have in HBase and needs to remain active for business reasons, you can periodically BulkLoad data from the system into HBase so that you can perform operations on it without impacting the system.

#### Using BulkLoad On A Secure Cluster

If you use security, HBase allows you to securely BulkLoad data into HBase. For a full explanation of how secure BulkLoad works, see HBase Transparent Encryption at Rest.

First, configure a `hbase.bulkload.staging.dir` which will be managed by HBase and whose subdirectories will be writable (but not readable) by HBase users. Next, add the `org.apache.hadoop.hbase.security.access.SecureBulkLoadEndpoint` coprocessor to your configuration, so that users besides the `hbase` user can BulkLoad files into HBase.

```
<property>
  <name>hbase.bulkload.staging.dir</name>
  <value>/tmp/hbase-staging</value>
</property>
<property>
  <name>hbase.coprocessor.region.classes</name>
  <value>org.apache.hadoop.hbase.security.access.SecureBulkLoadEndpoint</value>
</property>
```

#### More Information about BulkLoad

For more information and examples, as well as an explanation of the ImportTsv utility, which can be used to import data in text-delimited formats such as CSV, see this post on the Cloudera Blog.

## Using Cluster Replication

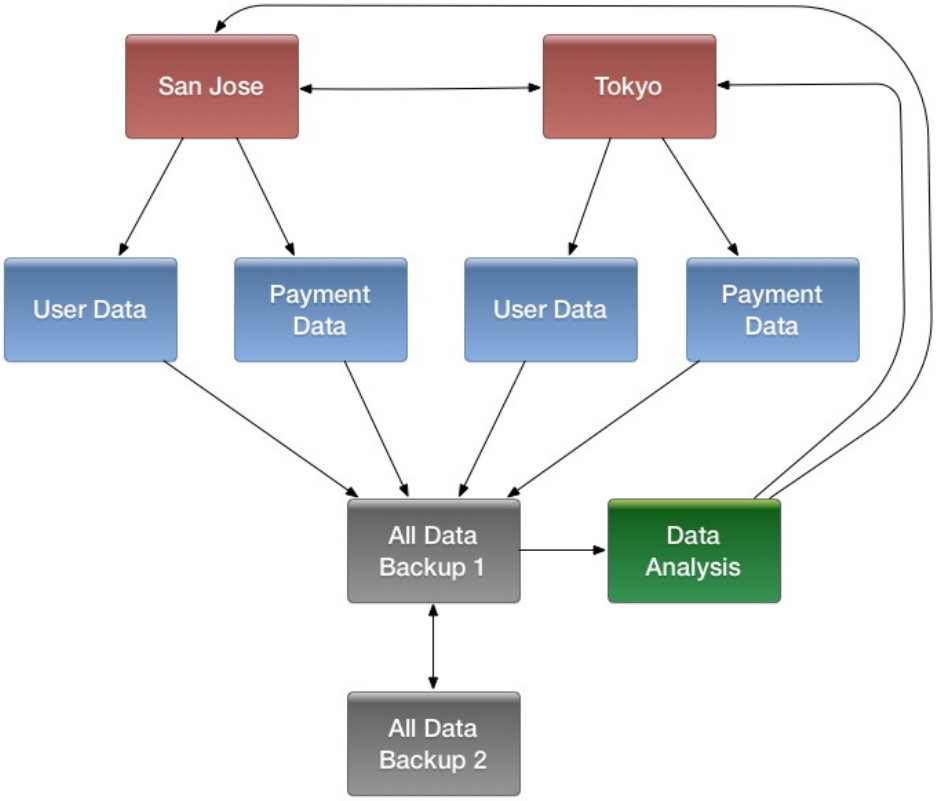
If your data is already in an HBase cluster, replication is useful for getting the data into additional HBase clusters. In HBase, cluster replication refers to keeping one cluster state synchronized with that of another cluster, using the write-ahead log (WAL) of the source cluster to propagate the changes. Replication is enabled at column family granularity. Before enabling replication for a column family, create the table and all column families to be replicated, on the destination cluster. Replication is supported both from CDH 5 to CDH 6 and from CDH 6 to CDH 5, the source and destination cluster do not have to run the same major version of CDH.

Cluster replication uses an active-push methodology. An HBase cluster can be a source (also called *active*, meaning that it writes new data), a destination (also called *passive*, meaning that it receives data using replication), or can fulfill both roles at once. Replication is asynchronous, and the goal of replication is consistency.

When data is replicated from one cluster to another, the original source of the data is tracked with a cluster ID, which is part of the metadata. All clusters that have already consumed the data are also tracked. This prevents replication loops.

**Common Replication Topologies**

- A central source cluster might propagate changes to multiple destination clusters, for failover or due to geographic distribution.
- A source cluster might push changes to a destination cluster, which might also push its own changes back to the original cluster.
- Many different low-latency clusters might push changes to one centralized cluster for backup or resource-intensive data-analytics jobs. The processed data might then be replicated back to the low-latency clusters.
- Multiple levels of replication can be chained together to suit your needs. The following diagram shows a hypothetical scenario. Use the arrows to follow the data paths.




At the top of the diagram, the `San Jose` and `Tokyo` clusters, shown in red, replicate changes to each other, and each also replicates changes to a `User Data` and a `Payment Data` cluster.

Each cluster in the second row, shown in blue, replicates its changes to the `All Data Backup 1` cluster, shown in grey. The `All Data Backup 1` cluster replicates changes to the `All Data Backup 2` cluster (also shown in grey), as well as the `Data Analysis` cluster (shown in green). `All Data Backup 2` also propagates any of its own changes back to `All Data Backup 1`.

The `Data Analysis` cluster runs MapReduce jobs on its data, and then pushes the processed data back to the `San Jose` and `Tokyo` clusters.

**Configuring Clusters for Replication**

To configure your clusters for replication, see HBase Replication and Configuring Secure HBase Replication. The following is a high-level overview of the steps to enable replication.

**Important:** You cannot run replication-related HBase commands as an HBase administrator. To run replication-related HBase commands, you must have HBase user permissions. If ZooKeeper uses Kerberos, configure HBase Shell to authenticate to ZooKeeper using Kerberos before attempting to run replication-related commands. No replication-related ACLs are available at this time.

1. Configure and start the source and destination clusters.
2. Create tables with the same names and column families on both the source and destination clusters, so that the destination cluster knows where to store data it receives. All hosts in the source and destination clusters should be reachable to each other. See Creating the Empty Table On the Destination Cluster.
3. On the source cluster, enable replication in Cloudera Manager, or by setting `hbase.replication` to `true` in `hbase-site.xml`.
4. Obtain Kerberos credentials as the HBase principal. Substitute your `fully.qualified.domain.name` and realm in the following command:

```
kinit -k -t /etc/hbase/conf/hbase.keytab hbase/fully.qualified.domain.name@YOUR-REALM.COM
```

5. On the source cluster, in HBase Shell, add the destination cluster as a peer, using the `add_peer` command. The syntax is as follows:

```
add_peer 'ID', 'CLUSTER_KEY'
```

   The ID must be a short integer. To compose the `CLUSTER_KEY`, use the following template:

```
hbase.zookeeper.quorum:hbase.zookeeper.property.clientPort:zookeeper.znode.parent
```

   If both clusters use the same ZooKeeper cluster, you must use a different **zookeeper.znode.parent**, because they cannot write in the same folder.

6. On the source cluster, configure each column family to be replicated by setting its `REPLICATION_SCOPE` to 1, using commands such as the following in HBase Shell.

```
hbase> disable 'example_table'
hbase> alter 'example_table', {NAME => 'example_family', REPLICATION_SCOPE => '1'}
hbase> enable 'example_table'
```

7. Verify that replication is occurring by examining the logs on the source cluster for messages such as the following.

```
Considering 1 rs, with ratio 0.1
Getting 1 rs from peer cluster # 0
Choosing peer 192.0.2.49:62020
```

8. To verify the validity of replicated data, use the included `VerifyReplication` MapReduce job on the source cluster, providing it with the ID of the replication peer and table name to verify. Other options are available, such as a time range or specific families to verify.

   The command has the following form:

```
hbase org.apache.hadoop.hbase.mapreduce.replication.VerifyReplication
[--starttime=timestamp1] [--stoptime=timestamp] [--families=comma separated list of
families] <peerId> <tablename>
```

   The `VerifyReplication` command prints `GOODROWS` and `BADROWS` counters to indicate rows that did and did not replicate correctly.

**Note:**

Some changes are not replicated and must be propagated by other means, such as Snapshots or CopyTable. See Initiating Replication When Data Already Exists for more details.

- Data that existed in the master before replication was enabled.
- Operations that bypass the WAL, such as when using BulkLoad or API calls such as `writeToWal(false)`.
- Table schema modifications.

## Using Pig and HCatalog

Apache Pig is a platform for analyzing large data sets using a high-level language. Apache HCatalog is a sub-project of Apache Hive, which enables reading and writing of data from one Hadoop utility to another. You can use a combination of Pig and HCatalog to import data into HBase. The initial format of your data and other details about your infrastructure determine the steps you follow to accomplish this task. The following simple example assumes that you can get your data into a TSV (text-separated value) format, such as a tab-delimited or comma-delimited text file.

1. Format the data as a TSV file. You can work with other file formats; see the Pig and HCatalog project documentation for more details.

   The following example shows a subset of data from Google's NGram Dataset, which shows the frequency of specific phrases or letter-groupings found in publications indexed by Google. Here, the first column has been added to this dataset as the row ID. The first column is formulated by combining the n-gram itself (in this case, `Zones`) with the line number of the file in which it occurs (`z_LINE_NUM`). This creates a format such as "`Zones_z_6230867`." The second column is the n-gram itself, the third column is the year of occurrence, the fourth column is the frequency of occurrence of that Ngram in that year, and the fifth column is the number of distinct publications. This extract is from the z file of the 1-gram dataset from version 20120701. The data is truncated at the `...` mark, for the sake of readability of this document. In most real-world scenarios, you will not work with tables that have five columns. Most HBase tables have one or two columns.

```
Zones_z_6230867 Zones 1507 1 1
Zones_z_6230868 Zones 1638 1 1
Zones_z_6230869 Zones 1656 2 1
Zones_z_6230870 Zones 1681 8 2
...
Zones_z_6231150 Zones 1996 17868 4356
Zones_z_6231151 Zones 1997 21296 4675
Zones_z_6231152 Zones 1998 20365 4972
Zones_z_6231153 Zones 1999 20288 5021
Zones_z_6231154 Zones 2000 22996 5714
Zones_z_6231155 Zones 2001 20469 5470
Zones_z_6231156 Zones 2002 21338 5946
Zones_z_6231157 Zones 2003 29724 6446
Zones_z_6231158 Zones 2004 23334 6524
Zones_z_6231159 Zones 2005 24300 6580
Zones_z_6231160 Zones 2006 22362 6707
Zones_z_6231161 Zones 2007 22101 6798
Zones_z_6231162 Zones 2008 21037 6328
```

2. Using the `hadoop fs` command, put the data into HDFS. This example places the file into an `/imported_data/` directory.

```
hadoop fs -put zones_frequency.tsv /imported_data/
```

3. Create and register a new HBase table in HCatalog, using the `hcat` command, passing it a DDL file to represent your table. You could also register an existing HBase table, using the same command. The DDL file format is specified as part of the Hive REST API. The following example illustrates the basic mechanism.

```
CREATE TABLE
zones_frequency_table (id STRING, ngram STRING, year STRING, freq STRING, sources STRING)
STORED BY 'org.apache.hcatalog.hbase.HBaseHCatStorageHandler'
TBLPROPERTIES (
  'hbase.table.name' = 'zones_frequency_table',
  'hbase.columns.mapping' = 'd:ngram,d:year,d:freq,d:sources',
  'hcat.hbase.output.bulkMode' = 'true'
);
```

```
hcat -f zones_frequency_table.ddl
```

4. Create a Pig file to process the TSV file created in step 1, using the DDL file created in step 3. Modify the file names and other parameters in this command to match your values if you use data different from this working example. `USING PigStorage('\t')` indicates that the input file is tab-delimited. For more details about Pig syntax, see the Pig Latin reference documentation.

```
A = LOAD 'hdfs:///imported_data/zones_frequency.tsv' USING PigStorage('\t') AS
(id:chararray, ngram:chararray, year:chararray, freq:chararray, sources:chararray);
-- DUMP A;
STORE A INTO 'zones_frequency_table' USING org.apache.hcatalog.pig.HCatStorer();
```

Save the file as `zones.bulkload.pig`.

5. Use the `pig` command to bulk-load the data into HBase.

```
pig -useHCatalog zones.bulkload.pig
```

The data is now in HBase and is available to use.

## Using the Java API

The Java API is the most common mechanism for getting data into HBase, through Put operations. The Thrift and REST APIs, as well as the HBase Shell, use the Java API. The following simple example ouses the Java API to put data into an HBase table. The Java API traverses the entire write path and can cause compactions and region splits, which can adversely affect performance.

```
...
  HTable table = null;
  try {
    table = myCode.createTable(tableName, fam);
    int i = 1;
    List<Put> puts = new ArrayList<Put>();
    for (String labelExp : labelExps) {
      Put put = new Put(Bytes.toBytes("row" + i));
      put.add(fam, qual, HConstants.LATEST_TIMESTAMP, value);
      puts.add(put);
      i++;
    }
    table.put(puts);
  } finally {
    if (table != null) {
      table.flushCommits();
    }
  }
...
```

## Using the Apache Thrift Proxy API

The Apache Thrift library provides cross-language client-server remote procedure calls (RPCs), using *Thrift bindings*. A Thrift binding is client code generated by the Apache Thrift Compiler for a target language (such as Python) that allows

communication between the Thrift server and clients using that client code. HBase includes an Apache Thrift Proxy API, which allows you to write HBase applications in Python, C, C++, or another language that Thrift supports. The Thrift Proxy API is slower than the Java API and may have fewer features. T use the Thrift Proxy API, you need to configure and run the HBase Thrift server on your cluster. You also need to install the Apache Thrift compiler on your development system.

After the Thrift server is configured and running, generate Thrift bindings for the language of your choice, using an IDL file. A HBase IDL file named `HBase.thrift` is included as part of HBase. After generating the bindings, copy the Thrift libraries for your language into the same directory as the generated bindings. In the following Python example, these libraries provide the `thrift.transport` and `thrift.protocol` libraries. These commands show how you might generate the Thrift bindings for Python and copy the libraries on a Linux system.

```
mkdir HBaseThrift
cd HBaseThrift/
thrift -gen py /path/to/Hbase.thrift
mv gen-py/* .
rm -rf gen-py/
mkdir thrift
cp -rp ~/Downloads/thrift-0.9.0/lib/py/src/* ./thrift/
```

The following example shows a simple Python application using the Thrift Proxy API.

```
from thrift.transport import TSocket
from thrift.protocol import TBinaryProtocol
from thrift.transport import TTransport
from hbase import Hbase

# Connect to HBase Thrift server
transport = TTransport.TBufferedTransport(TSocket.TSocket(host, port))
protocol = TBinaryProtocol.TBinaryProtocolAccelerated(transport)

# Create and open the client connection
client = Hbase.Client(protocol)
transport.open()

# Modify a single row
mutations = [Hbase.Mutation(
  column='columnfamily:columndescriptor', value='columnvalue')]
client.mutateRow('tablename', 'rowkey', mutations)

# Modify a batch of rows
# Create a list of mutations per work of Shakespeare
mutationsbatch = []

for line in myDataFile:
    rowkey = username + "-" + filename + "-" + str(linenumber).zfill(6)

    mutations = [
            Hbase.Mutation(column=messagecolumncf, value=line.strip()),
            Hbase.Mutation(column=linenumbercolumncf, value=encode(linenumber)),
            Hbase.Mutation(column=usernamecolumncf, value=username)
        ]

       mutationsbatch.append(Hbase.BatchMutation(row=rowkey,mutations=mutations))

# Run the mutations for all the lines in myDataFile
client.mutateRows(tablename, mutationsbatch)

transport.close()
```

The Thrift Proxy API does not support writing to HBase clusters that are secured using Kerberos.

This example was modified from the following two blog posts on http://www.cloudera.com. See them for more details.

- Using the HBase Thrift Interface, Part 1
- Using the HBase Thrift Interface, Part 2

## Using the REST Proxy API

After configuring and starting HBase on your cluster, you can use the HBase REST Proxy API to stream data into HBase, from within another application or shell script, or by using an HTTP client such as `wget` or `curl`. The REST Proxy API is slower than the Java API and may have fewer features. This approach is simple and does not require advanced development experience to implement. However, like the Java and Thrift Proxy APIs, it uses the full write path and can cause compactions and region splits.

Specified addresses without existing data create new values. Specified addresses with existing data create new versions, overwriting an existing version if the row, column:qualifier, and timestamp all match that of the existing value.

```
curl -H "Content-Type: text/xml" http://localhost:8000/test/testrow/test:testcolumn
```

The REST Proxy API does not support writing to HBase clusters that are secured using Kerberos.

For full documentation and more examples, see the REST Proxy API documentation.

## Using Flume

Apache Flume is a fault-tolerant system designed for ingesting data into HDFS, for use with Hadoop. You can configure Flume to write data directly into HBase. Flume includes a sink designed to work with HBase: HBase2Sink (org.apache.flume.sink.hbase2.HBase2Sink). HBase2Sink supports HBase 2 IPC calls, and allows you to write data to an HBase cluster that is secured by Kerberos.

The following code is an example configuration for HBase2Sink. For more information about configuring HBase2Sink, see the Flume documentation. The `table`, `columnFamily`, and `column` parameters correlate to the HBase table, column family, and column where the data is to be imported. The serializer is the class that converts the data at the source into something HBase can use. Configure your sinks in the Flume configuration file.

In practice, you usually need to write your own serializer, which implements HBase2EventSerializer. The HBase2EventSerializer converts Flume Events into one or more HBase Puts, sends them to the HBase cluster, and is closed when the HBase2Sink stops.

**HBase2Sink**:

```
#Use the HBase2Sink
host1.sinks.sink1.type = org.apache.flume.sink.hbase2.HBase2Sink
host1.sinks.sink1.channel = ch1
host1.sinks.sink1.table = transactions
host1.sinks.sink1.columnFamily = clients
host1.sinks.sink1.column = charges
host1.sinks.sink1.batchSize = 5000
#Use the SimpleHBase2EventSerializer that comes with Flume
host1.sinks.sink1.serializer = org.apache.flume.sink.hbase2.SimpleHBase2EventSerializer
host1.sinks.sink1.serializer.incrementColumn = icol
host1.channels.ch1.type=memory
```

The following serializer, based on an Apache Flume blog post by Dan Sandler, splits the event body based on a delimiter and inserts each split into a different column. The column names are defined in the serializer.columns sink parameter. The row key is defined in the event header. When each event is received, a counter is incremented to track the number of events received.

```
import org.apache.flume.Context;
import org.apache.flume.Event;
import org.apache.flume.FlumeException;
import org.apache.flume.conf.ComponentConfiguration;
import org.apache.flume.sink.hbase2.HBase2EventSerializer;
import org.apache.hadoop.hbase.client.Increment;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.client.Row;

import java.util.ArrayList;
import java.util.List;

/**
```

```
 * A serializer for the HBase2Sink, which splits the event body into
 * multiple columns and inserts them into a row whose key is available in
 * the headers
 */
public class SplittingSerializer implements HBase2EventSerializer {
    private byte[] colFam;
    private Event currentEvent;
    private byte[][] columnNames;
    private final List<Row> puts = new ArrayList<>();
    private final List<Increment> incs = new ArrayList<>();
    private byte[] currentRowKey;
    private final byte[] eventCountCol = "eventCount".getBytes();

    @Override
    public void initialize(Event event, byte[] cf) {
        this.currentEvent = event;
        String rowKeyStr = currentEvent.getHeaders().get("rowKey");
        if (rowKeyStr == null) {
            throw new FlumeException("No row key found in headers!");
        }
        currentRowKey = rowKeyStr.getBytes();

        this.colFam = cf;
    }

    @Override
    public List<Row> getActions() {
        // Split the event body and get the values for the columns
        String eventStr = new String(currentEvent.getBody());
        String[] cols = eventStr.split(",");
        puts.clear();
        for (int i = 0; i < cols.length; i++) {
            //Generate a Put for each column.
            Put put = new Put(currentRowKey);
            put.addColumn(colFam, columnNames[i], cols[i].getBytes());
            puts.add(put);
        }
        return puts;
    }

    @Override
    public List<Increment> getIncrements() {
        incs.clear();
        //Increment the number of events received
        Increment inc = new Increment("totalEvents".getBytes());
        inc.addColumn(colFam, eventCountCol, 1);
        incs.add(inc);
        return incs;
    }

    @Override
    public void close() {
        colFam = null;
        currentEvent = null;
        columnNames = null;
        currentRowKey = null;
    }

    @Override
    public void configure(Context context) {
        //Get the column names from the configuration
        String cols = context.getString("columns");
        String[] names = cols.split(",");
        byte[][] columnNames = new byte[names.length][];
        int i = 0;
        for (String name : names) {
            columnNames[i++] = name.getBytes();
        }
        this.columnNames = columnNames;
    }

    @Override
    public void configure(ComponentConfiguration conf) {
```

```
    }
}
```

## Using Sqoop

Sqoop can import records into a table in HBase. It has an out-of-the-box support for HBase.

There are two mandatory options you must specify when using the `sqoop import` command to import data into HBase using Sqoop:

- `--hbase-table`: Specifies the name of the table in HBase to which you want to import your data.
- `--column-family`: Specifies into which column family Sqoop imports the data of your tables.

For example, you can import the table *cities* into an already existing HBase table with the same name and use the column family name *world*:

```
sqoop import --connect jdbc:mysql://mysql.example.com/sqoop --username sqoop --password
 sqoop --table cities --hbase-table cities --column-family world
```

If the target table and column family do not exist, the Sqoop job will exit with an error. You must create the target table and column family before running an import. If you specify `--hbase-create-table`, Sqoop creates the target table and column family if they do not exist, using the default parameters from your HBase configuration.

Sqoop needs to identify which RDBMS column is used as row key column in the HBase table. There are three ways to do this:

- By default, with the column name specified in the `--split-by` option
- With the primary key of the table, if it is available
- With the `--hbase-row-key` parameter, which overrides both the `--split-by` option and the primary key of the table

For more information on data insertion into HBase, see Sqoop User Guide.

### Import NULL Column Updates into HBase

You can specify how Sqoop handles RDBMS table column updated to NULL during incremental import.

There are two modes for this, ignore and delete. You can specify the mode using the `--hbase-null-incrementel-mode` option:

- `-ignore`: This is the default value. If the source table's column is updated to NULL, the target HBase table will still show the previous value for that column.
- `-delete`: If the source table's column is updated to NULL, all previous versions of the column will be deleted from HBase. When checking the column in HBase using the Java API, a null value will be displayed.

**Examples:**

Execute an incremental import to an HBase table and ignore the columns which were updated to NULL in the relational database:

```
sqoop import --connect $CONN --username $USER --password $PASS --table "hbase_test"
--hbase-table hbase_test --column-family data -m 1 --incremental lastmodified
--check-column date_modified --last-value "2017-12-15 10:58:44.0" --merge-key id
--hbase-null-incremental-mode ignore
```

Execute an incremental import to an HBase table and delete all the versions of the columns which were updated to NULL in the relational database:

```
sqoop import --connect $CONN --username $USER --password $PASS --table "hbase_test"
--hbase-table hbase_test --column-family data -m 1 --incremental lastmodified
```

```
--check-column date_modified --last-value "2017-12-15 10:58:44.0" --merge-key id
--hbase-null-incremental-mode delete
```

## Using Spark

For instructions on configuring an HBase service as a Spark service dependency, see Accessing HBase from Spark.

You can write data to HBase from Apache Spark by using `def saveAsHadoopDataset(conf: JobConf): Unit`. This example is adapted from a post on the spark-users mailing list.

```
// Note: mapred package is used, instead of the
// mapreduce package which contains new hadoop APIs.

import org.apache.hadoop.hbase.mapred.TableOutputFormat
import org.apache.hadoop.hbase.client
// ... some other settings

val conf = HBaseConfiguration.create()

// general hbase settings
conf.set("hbase.rootdir",
         "hdfs://" + nameNodeURL + ":" + hdfsPort + "/hbase")
conf.setBoolean("hbase.cluster.distributed", true)
conf.set("hbase.zookeeper.quorum", hostname)
conf.setInt("hbase.client.scanner.caching", 10000)
// ... some other settings

val jobConfig: JobConf = new JobConf(conf, this.getClass)

// Note:  TableOutputFormat is used as deprecated code
// because JobConf is an old hadoop API
jobConfig.setOutputFormat(classOf[TableOutputFormat])
jobConfig.set(TableOutputFormat.OUTPUT_TABLE, outputTable)
```

Next, provide the mapping between how the data looks in Spark and how it should look in HBase. The following example assumes that your HBase table has two column families, col_1 and col_2, and that your data is formatted in sets of three in Spark, like (row_key, col_1, col_2).

```
def convert(triple: (Int, Int, Int)) = {
      val p = new Put(Bytes.toBytes(triple._1))
      p.add(Bytes.toBytes("cf"),
            Bytes.toBytes("col_1"),
            Bytes.toBytes(triple._2))
      p.add(Bytes.toBytes("cf"),
            Bytes.toBytes("col_2"),
            Bytes.toBytes(triple._3))
      (new ImmutableBytesWritable, p)
}
```

To write the data from Spark to HBase, you might use:

```
new PairRDDFunctions(localData.map(convert)).saveAsHadoopDataset(jobConfig)
```

## Using Spark and Kafka

For instructions on configuring an HBase service as a Spark service dependency, see Accessing HBase from Spark.

This example, written in Scala, uses Apache Spark in conjunction with the Apache Kafka message bus to stream data from Spark to HBase. The example was provided in SPARK-944. It produces some random words and then stores them in an HBase table, creating the table if necessary.

```
package org.apache.spark.streaming.examples

import java.util.Properties

import kafka.producer._
```

```
import org.apache.hadoop.hbase.{ HBaseConfiguration, HColumnDescriptor, HTableDescriptor
 }
import org.apache.hadoop.hbase.client.{ HBaseAdmin, Put }
import org.apache.hadoop.hbase.io.ImmutableBytesWritable
import org.apache.hadoop.hbase.mapred.TableOutputFormat
import org.apache.hadoop.hbase.mapreduce.TableInputFormat
import org.apache.hadoop.hbase.util.Bytes
import org.apache.hadoop.mapred.JobConf
import org.apache.spark.SparkContext
import org.apache.spark.rdd.{ PairRDDFunctions, RDD }
import org.apache.spark.streaming._
import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.streaming.kafka._

object MetricAggregatorHBase {
  def main(args : Array[String]) {
    if (args.length < 6) {
     System.err.println("Usage: MetricAggregatorTest <master> <zkQuorum> <group> <topics>
 <destHBaseTableName> <numThreads>")
      System.exit(1)
    }

    val Array(master, zkQuorum, group, topics, hbaseTableName, numThreads) = args

    val conf = HBaseConfiguration.create()
    conf.set("hbase.zookeeper.quorum", zkQuorum)

    // Initialize hBase table if necessary
    val admin = new HBaseAdmin(conf)
    if (!admin.isTableAvailable(hbaseTableName)) {
      val tableDesc = new HTableDescriptor(hbaseTableName)
      tableDesc.addFamily(new HColumnDescriptor("metric"))
      admin.createTable(tableDesc)
    }

    // setup streaming context
    val ssc = new StreamingContext(master, "MetricAggregatorTest", Seconds(2),
      System.getenv("SPARK_HOME"), StreamingContext.jarOfClass(this.getClass))
    ssc.checkpoint("checkpoint")

    val topicpMap = topics.split(",").map((_, numThreads.toInt)).toMap
    val lines = KafkaUtils.createStream(ssc, zkQuorum, group, topicpMap)
      .map { case (key, value) => ((key, Math.floor(System.currentTimeMillis() /
60000).toLong * 60), value.toInt) }

    val aggr = lines.reduceByKeyAndWindow(add _, Minutes(1), Minutes(1), 2)

    aggr.foreach(line => saveToHBase(line, zkQuorum, hbaseTableName))

    ssc.start

    ssc.awaitTermination
  }

  def add(a : Int, b : Int) = { (a + b) }

  def saveToHBase(rdd : RDD[((String, Long), Int)], zkQuorum : String, tableName :
String) = {
    val conf = HBaseConfiguration.create()
    conf.set("hbase.zookeeper.quorum", zkQuorum)

    val jobConfig = new JobConf(conf)
    jobConfig.set(TableOutputFormat.OUTPUT_TABLE, tableName)
    jobConfig.setOutputFormat(classOf[TableOutputFormat])

    new PairRDDFunctions(rdd.map { case ((metricId, timestamp), value) =>
createHBaseRow(metricId, timestamp, value) }).saveAsHadoopDataset(jobConfig)
  }

  def createHBaseRow(metricId : String, timestamp : Long, value : Int) = {
    val record = new Put(Bytes.toBytes(metricId + "~" + timestamp))
```

```
    record.add(Bytes.toBytes("metric"), Bytes.toBytes("col"),
Bytes.toBytes(value.toString))

    (new ImmutableBytesWritable, record)
  }

}

// Produces some random words between 1 and 100.
object MetricDataProducer {

  def main(args : Array[String]) {
    if (args.length < 2) {
      System.err.println("Usage: MetricDataProducer <metadataBrokerList> <topic>
<messagesPerSec>")
      System.exit(1)
    }

    val Array(brokers, topic, messagesPerSec) = args

    // ZooKeeper connection properties
    val props = new Properties()
    props.put("metadata.broker.list", brokers)
    props.put("serializer.class", "kafka.serializer.StringEncoder")

    val config = new ProducerConfig(props)
    val producer = new Producer[String, String](config)

    // Send some messages
    while (true) {
      val messages = (1 to messagesPerSec.toInt).map { messageNum =>
        {
          val metricId = scala.util.Random.nextInt(10)
          val value = scala.util.Random.nextInt(1000)
          new KeyedMessage[String, String](topic, metricId.toString, value.toString)
        }
      }.toArray

      producer.send(messages : _*)
      Thread.sleep(100)
    }
  }
}
```

### Using a Custom MapReduce Job

Many of the methods to import data into HBase use MapReduce implicitly. If none of those approaches fit your needs, you can use MapReduce directly to convert data to a series of HFiles or API calls for import into HBase. In this way, you can import data from Avro, Parquet, or another format into HBase, or export data from HBase into another format, using API calls such as `TableOutputFormat`, `HFileOutputFormat`, and `TableInputFormat`.

## Reading Data from HBase

`Get` and `Scan` are the two ways to read data from HBase, aside from manually parsing HFiles. A `Get` is simply a `Scan` limited by the API to one row. A `Scan` fetches zero or more rows of a table. By default, a `Scan` reads the entire table from start to end. You can limit your `Scan` results in several different ways, which affect the `Scan`'s load in terms of IO, network, or both, as well as processing load on the client side. This topic is provided as a quick reference. Refer to the API documentation for Scan for more in-depth information. You can also perform Gets and Scan using the HBase Shell, the REST API, or the Thrift API.

- Specify a `startrow` or `stoprow` or both. Neither `startrow` nor `stoprow` need to exist. Because HBase sorts rows lexicographically, it will return the first row after `startrow` would have occurred, and will stop returning rows after `stoprow` would have occurred.The goal is to reduce IO and network.
  - The `startrow` is inclusive and the `stoprow` is exclusive. Given a table with rows a, b, c, d, e, `f`, and `startrow` of c and `stoprow` of `f`, rows `c-e` are returned.

  - If you omit `startrow`, the first row of the table is the `startrow`.
  - If you omit the `stoprow`, all results after `startrow` (including `startrow`) are returned.
  - If `startrow` is lexicographically after `stoprow`, and you set `Scan setReversed(boolean reversed)` to `true`, the results are returned in reverse order. Given the same table above, with rows `a-f`, if you specify `c` as the stoprow and `f` as the startrow, rows `f`, `e`, and `d` are returned.

```
Scan()
Scan(byte[] startRow)
Scan(byte[] startRow, byte[] stopRow)
```

- Specify a scanner cache that will be filled before the Scan result is returned, setting `setCaching` to the number of rows to cache before returning the result. By default, the caching setting on the table is used. The goal is to balance IO and network load.

```
public Scan setCaching(int caching)
```

- To limit the number of columns if your table has very wide rows (rows with a large number of columns), use setBatch(int batch) and set it to the number of columns you want to return in one batch. A large number of columns is not a recommended design pattern.

```
public Scan setBatch(int batch)
```

- To specify a maximum result size, use `setMaxResultSize(long)`, with the number of bytes. The goal is to reduce IO and network.

```
public Scan setMaxResultSize(long maxResultSize)
```

- When you use `setCaching` and `setMaxResultSize` together, single server requests are limited by either number of rows or maximum result size, whichever limit comes first.
- You can limit the scan to specific column families or columns by using `addFamily` or `addColumn`. The goal is to reduce IO and network. IO is reduced because each column family is represented by a Store on each RegionServer, and only the Stores representing the specific column families in question need to be accessed.

```
public Scan addColumn(byte[] family,
            byte[] qualifier)

public Scan addFamily(byte[] family)
```

- You can specify a range of timestamps or a single timestamp by specifying setTimeRange or setTimestamp.

```
public Scan setTimeRange(long minStamp,
                long maxStamp)
                  throws IOException

public Scan setTimeStamp(long timestamp)
                  throws IOException
```

- You can retrieve a maximum number of versions by using setMaxVersions.

```
public Scan setMaxVersions(int maxVersions)
```

- You can use a filter by using `setFilter`. Filters are discussed in detail in HBase Filtering on page 79 and the Filter API.

```
public Scan setFilter(Filter filter)
```

- You can disable the server-side block cache for a specific scan using the API `setCacheBlocks(boolean)`. This is an expert setting and should only be used if you know what you are doing.

### Perform Scans Using HBase Shell

You can perform scans using HBase Shell, for testing or quick queries. Use the following guidelines or issue the scan command in HBase Shell with no parameters for more usage information. This represents only a subset of possibilities.

```
# Display usage information
hbase> scan

# Scan all rows of table 't1'
hbase> scan 't1'

# Specify a startrow, limit the result to 10 rows, and only return selected columns
hbase> scan 't1', {COLUMNS => ['c1', 'c2'], LIMIT => 10, STARTROW => 'xyz'}

# Specify a timerange
hbase> scan 't1', {TIMERANGE => [1303668804, 1303668904]}

# Specify a custom filter
hbase> scan 't1', {FILTER => org.apache.hadoop.hbase.filter.ColumnPaginationFilter.new(1,
 0)}

# Specify a row prefix filter and another custom filter
hbase> scan 't1', {ROWPREFIXFILTER => 'row2',
                        FILTER => (QualifierFilter (>=, 'binary:xyz')) AND
(TimestampsFilter ( 123, 456))}

# Disable the block cache for a specific scan (experts only)
hbase> scan 't1', {COLUMNS => ['c1', 'c2'], CACHE_BLOCKS => false}
```

## Hedged Reads

Hadoop 2.4 introduced a new feature called *hedged reads*. If a read from a block is slow, the HDFS client starts up another parallel, 'hedged' read against a different block replica. The result of whichever read returns first is used, and the outstanding read is cancelled. This feature helps in situations where a read occasionally takes a long time rather than when there is a systemic problem. Hedged reads can be enabled for HBase when the HFiles are stored in HDFS. This feature is disabled by default.

### Enabling Hedged Reads for HBase

**Minimum Required Role:** Configurator (also provided by **Cluster Administrator, Full Administrator**)

1. Go to the HBase service.
2. Click the **Configuration** tab.
3. Select **Scope** > **HBASE-1 (Service-Wide)**.
4. Select **Category** > **Performance**.
5. Configure the **HDFS Hedged Read Threadpool Size** and **HDFS Hedged Read Delay Threshold** properties. The descriptions for each of these properties on the configuration pages provide more information.
6. Enter a **Reason for change**, and then click **Save Changes** to commit the changes.

# Hedged Reads

Hadoop 2.4 introduced a new feature called *hedged reads*. If a read from a block is slow, the HDFS client starts up another parallel, 'hedged' read against a different block replica. The result of whichever read returns first is used, and the outstanding read is cancelled. This feature helps in situations where a read occasionally takes a long time rather than when there is a systemic problem. Hedged reads can be enabled for HBase when the HFiles are stored in HDFS. This feature is disabled by default.

## Enabling Hedged Reads for HBase

**Minimum Required Role:** Configurator (also provided by **Cluster Administrator, Full Administrator**)

1. Go to the HBase service.

2. Click the **Configuration** tab.
3. Select **Scope** > **HBASE-1 (Service-Wide)**.
4. Select **Category** > **Performance**.
5. Configure the **HDFS Hedged Read Threadpool Size** and **HDFS Hedged Read Delay Threshold** properties. The descriptions for each of these properties on the configuration pages provide more information.
6. Enter a **Reason for change**, and then click **Save Changes** to commit the changes.

### Monitoring the Performance of Hedged Reads

You can monitor the performance of hedged reads using the following metrics emitted by Hadoop when hedged reads are enabled.

- **hedgedReadOps** - the number of hedged reads that have occurred
- **hedgeReadOpsWin** - the number of times the hedged read returned faster than the original read

## HBase Filtering

When reading data from HBase using Get or Scan operations, you can use custom filters to return a subset of results to the client. While this does not reduce server-side IO, it does reduce network bandwidth and reduces the amount of data the client needs to process. Filters are generally used using the Java API, but can be used from HBase Shell for testing and debugging purposes.

For more information on Gets and Scans in HBase, see Reading Data from HBase on page 76.

#### Dynamically Loading a Custom Filter

CDH, by default, has the ability to dynamically load a custom filter by adding a JAR with your filter to the directory specified by the `hbase.dynamic.jars.dir` property (which defaults to the `lib/` directory under the HBase root directory).

To disable automatic loading of dynamic JARs, set `hbase.use.dynamic.jars` to `false` in the advanced configuration snippet for `hbase-site.xml` if you use Cloudera Manager, or to `hbase-site.xml` otherwise.

#### Filter Syntax Guidelines

HBase filters take zero or more arguments, in parentheses. Where the argument is a string, it is surrounded by single quotes ('string').

#### Logical Operators, Comparison Operators and Comparators

Filters can be combined together with logical operators. Some filters take a combination of comparison operators and comparators. Following is the list of each.

**Logical Operators**

- AND - the key-value must pass both the filters to be included in the results.
- OR - the key-value must pass at least one of the filters to be included in the results.
- SKIP - for a particular row, if any of the key-values do not pass the filter condition, the entire row is skipped.
- WHILE - For a particular row, it continues to emit key-values until a key-value is reached that fails the filter condition.
- Compound Filters - Using these operators, a hierarchy of filters can be created. For example:

```
(Filter1 AND Filter2)OR(Filter3 AND Filter4)
```

**Comparison Operators**

- LESS (<)
- LESS_OR_EQUAL (<=)
- EQUAL (=)

- NOT_EQUAL (!=)
- GREATER_OR_EQUAL (>=)
- GREATER (>)
- NO_OP (no operation)

**Comparators**

- BinaryComparator - lexicographically compares against the specified byte array using the `Bytes.compareTo(byte[], byte[])` method.
- BinaryPrefixComparator - lexicographically compares against a specified byte array. It only compares up to the length of this byte array.
- RegexStringComparator - compares against the specified byte array using the given regular expression. Only `EQUAL` and `NOT_EQUAL` comparisons are valid with this comparator.
- SubStringComparator - tests whether or not the given substring appears in a specified byte array. The comparison is case insensitive. Only `EQUAL` and `NOT_EQUAL` comparisons are valid with this comparator.

**Examples**

```
Example1: >, 'binary:abc' will match everything that is lexicographically greater than
 "abc"
Example2: =, 'binaryprefix:abc' will match everything whose first 3 characters are
lexicographically equal to "abc"
Example3: !=, 'regexstring:ab*yz' will match everything that doesn't begin with "ab"
and ends with "yz"
Example4: =, 'substring:abc123' will match everything that begins with the substring
"abc123"
```

### Compound Operators

Within an expression, parentheses can be used to group clauses together, and parentheses have the highest order of precedence.

`SKIP` and `WHILE` operators are next, and have the same precedence.

The `AND` operator is next.

The `OR` operator is next.

**Examples**

```
A filter string of the form: “Filter1 AND Filter2 OR Filter3” will be evaluated as:
“(Filter1 AND Filter2) OR Filter3”
A filter string of the form: “Filter1 AND SKIP Filter2 OR Filter3” will be evaluated
as: “(Filter1 AND (SKIP Filter2)) OR Filter3”
```

### Filter Types

HBase includes several filter types, as well as the ability to group filters together and create your own custom filters.

- **KeyOnlyFilter** - takes no arguments. Returns the key portion of each key-value pair.

```
Syntax: KeyOnlyFilter ()
```

- **FirstKeyOnlyFilter** - takes no arguments. Returns the key portion of the first key-value pair.

```
Syntax: FirstKeyOnlyFilter ()
```

- **PrefixFilter** - takes a single argument, a prefix of a row key. It returns only those key-values present in a row that start with the specified row prefix

```
Syntax:  PrefixFilter (‘<row_prefix>’)

Example: PrefixFilter (‘Row’)
```

- **ColumnPrefixFilter** - takes a single argument, a column prefix. It returns only those key-values present in a column that starts with the specified column prefix.

```
Syntax:  ColumnPrefixFilter (‘<column_prefix>’)

Example: ColumnPrefixFilter (‘Col’)
```

- **MultipleColumnPrefixFilter** - takes a list of column prefixes. It returns key-values that are present in a column that starts with *any* of the specified column prefixes.

```
Syntax:  MultipleColumnPrefixFilter (‘<column_prefix>’, ‘<column_prefix>’, …, ‘<column_prefix>’)

Example: MultipleColumnPrefixFilter (‘Col1’, ‘Col2’)
```

- **ColumnCountGetFilter** - takes one argument, a limit. It returns the first `limit` number of columns in the table.

```
Syntax:  ColumnCountGetFilter (‘<limit>’)

Example: ColumnCountGetFilter (4)
```

- **PageFilter** - takes one argument, a page size. It returns page `size` number of rows from the table.

```
Syntax:  PageFilter (‘<page_size>’)

Example: PageFilter (2)
```

- **ColumnPaginationFilter** - takes two arguments, a limit and offset. It returns limit number of columns after offset number of columns. It does this for all the rows.

```
Syntax:  ColumnPaginationFilter (‘<limit>’, ‘<offset>’)

Example: ColumnPaginationFilter (3, 5)
```

- **InclusiveStopFilter** - takes one argument, a row key on which to stop scanning. It returns all key-values present in rows *up to and including* the specified row.

```
Syntax:  InclusiveStopFilter (‘<stop_row_key>’)

Example: InclusiveStopFilter (‘Row2’)
```

- **TimeStampsFilter** - takes a list of timestamps. It returns those key-values whose timestamps matches *any* of the specified timestamps.

```
Syntax:  TimeStampsFilter (<timestamp>, <timestamp>, ... ,<timestamp>)

Example: TimeStampsFilter (5985489, 48895495, 58489845945)
```

- **RowFilter** - takes a compare operator and a comparator. It compares each row key with the comparator using the compare operator and if the comparison returns `true`, it returns all the key-values in that row.

```
Syntax:  RowFilter (<compareOp>, ‘<row_comparator>’)

Example: RowFilter (<=, ‘binary:xyz)
```

- **FamilyFilter** - takes a compare operator and a comparator. It compares each family name with the comparator using the compare operator and if the comparison returns `true`, it returns all the key-values in that family.

```
Syntax:  FamilyFilter (<compareOp>, '<family_comparator>')

Example: FamilyFilter (>=, 'binaryprefix:FamilyB')
```

- **QualifierFilter** - takes a compare operator and a comparator. It compares each qualifier name with the comparator using the compare operator and if the comparison returns `true`, it returns all the key-values in that column.

```
Syntax:  QualifierFilter (<compareOp>, '<qualifier_comparator>')

Example: QualifierFilter (=, 'substring:Column1')
```

- **ValueFilter** - takes a compare operator and a comparator. It compares each value with the comparator using the compare operator and if the comparison returns `true`, it returns that key-value.

```
Syntax:  ValueFilter (<compareOp>, '<value_comparator>')

Example: ValueFilter (!=, 'binary:Value')
```

- **DependentColumnFilter** - takes two arguments required arguments, a family and a qualifier. It tries to locate this column in each row and returns all key-values in that row that have the same timestamp. If the row does not contain the specified column, none of the key-values in that row will be returned.

  The filter can also take an optional boolean argument, `dropDependentColumn`. If set to `true`, the column used for the filter does not get returned.

  The filter can also take two more additional optional arguments, a compare operator and a value comparator, which are further checks in addition to the family and qualifier. If the dependent column is found, its value should also pass the value check. If it does pass the value check, only then is its timestamp taken into consideration.

```
Syntax:  DependentColumnFilter ('<family>', '<qualifier>', <boolean>, <compare operator>,
 '<value comparator')
         DependentColumnFilter ('<family>', '<qualifier>', <boolean>)
         DependentColumnFilter ('<family>', '<qualifier>')

Example: DependentColumnFilter ('conf', 'blacklist', false, >=, 'zebra')
         DependentColumnFilter ('conf', 'blacklist', true)
         DependentColumnFilter ('conf', 'blacklist')
```

- **SingleColumnValueFilter** - takes a column family, a qualifier, a compare operator and a comparator. If the specified column is not found, all the columns of that row will be emitted. If the column is found and the comparison with the comparator returns `true`, all the columns of the row will be emitted. If the condition fails, the row will not be emitted.

  This filter also takes two additional optional boolean arguments, `filterIfColumnMissing` and `setLatestVersionOnly`.

  If the `filterIfColumnMissing` flag is set to `true`, the columns of the row will not be emitted if the specified column to check is not found in the row. The default value is `false`.

  If the `setLatestVersionOnly` flag is set to `false`, it will test previous versions (timestamps) in addition to the most recent. The default value is `true`.

  These flags are optional and dependent on each other. You must set neither or both of them together.

```
Syntax:  SingleColumnValueFilter ('<family>', '<qualifier>', <compare operator>,
'<comparator>', <filterIfColumnMissing_boolean>, <latest_version_boolean>)
Syntax:  SingleColumnValueFilter ('<family>', '<qualifier>', <compare operator>,
'<comparator>')

Example: SingleColumnValueFilter ('FamilyA', 'Column1', <=, 'abc', true, false)
Example: SingleColumnValueFilter ('FamilyA', 'Column1', <=, 'abc')
```

- **SingleColumnValueExcludeFilter** - takes the same arguments and behaves same as `SingleColumnValueFilter`. However, if the column is found and the condition passes, all the columns of the row will be emitted except for the tested column value.

```
Syntax:  SingleColumnValueExcludeFilter (<family>, <qualifier>, <compare operators>,
<comparator>, <latest_version_boolean>, <filterIfColumnMissing_boolean>)
Syntax:  SingleColumnValueExcludeFilter (<family>, <qualifier>, <compare operator>
<comparator>)

Example: SingleColumnValueExcludeFilter (‘FamilyA’, ‘Column1’, ‘<=’, ‘abc’, ‘false’,
‘true’)
Example: SingleColumnValueExcludeFilter (‘FamilyA’, ‘Column1’, ‘<=’, ‘abc’)
```

- **ColumnRangeFilter** - takes either `minColumn`, `maxColumn`, or both. Returns only those keys with columns that are between `minColumn` and `maxColumn`. It also takes two boolean variables to indicate whether to include the `minColumn` and `maxColumn` or not. If you don’t want to set the `minColumn` or the `maxColumn`, you can pass in an empty argument.

```
Syntax:  ColumnRangeFilter (‘<minColumn >’, <minColumnInclusive_bool>, ‘<maxColumn>’,
<maxColumnInclusive_bool>)

Example: ColumnRangeFilter (‘abc’, true, ‘xyz’, false)
```

- **Custom Filter** - You can create a custom filter by implementing the Filter class. The JAR must be available on all RegionServers.

### HBase Shell Example

This example scans the 'users' table for rows where the contents of the `cf:name` column equals the string 'abc'.

```
hbase> scan 'users', { FILTER => SingleColumnValueFilter.new(Bytes.toBytes('cf'),
      Bytes.toBytes('name'), CompareFilter::CompareOp.valueOf('EQUAL'),
      BinaryComparator.new(Bytes.toBytes('abc')))}
```

### Java API Example

This example, taken from the HBase unit test found in `hbase-server/src/test/java/org/apache/hadoop/hbase/filter/TestSingleColumnValueFilter.java`, shows how to use the Java API to implement several different filters..

```
/**
 *
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.apache.hadoop.hbase.filter;

import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

import java.util.regex.Pattern;
```

```
import org.apache.hadoop.hbase.KeyValue;
import org.apache.hadoop.hbase.SmallTests;
import org.apache.hadoop.hbase.filter.CompareFilter.CompareOp;
import org.apache.hadoop.hbase.util.Bytes;
import org.junit.Before;
import org.junit.Test;
import org.junit.experimental.categories.Category;

/**
 * Tests the value filter
 */
@Category(SmallTests.class)
public class TestSingleColumnValueFilter {
  private static final byte[] ROW = Bytes.toBytes("test");
  private static final byte[] COLUMN_FAMILY = Bytes.toBytes("test");
  private static final byte [] COLUMN_QUALIFIER = Bytes.toBytes("foo");
  private static final byte[] VAL_1 = Bytes.toBytes("a");
  private static final byte[] VAL_2 = Bytes.toBytes("ab");
  private static final byte[] VAL_3 = Bytes.toBytes("abc");
  private static final byte[] VAL_4 = Bytes.toBytes("abcd");
  private static final byte[] FULLSTRING_1 =
    Bytes.toBytes("The quick brown fox jumps over the lazy dog.");
  private static final byte[] FULLSTRING_2 =
    Bytes.toBytes("The slow grey fox trips over the lazy dog.");
  private static final String QUICK_SUBSTR = "quick";
  private static final String QUICK_REGEX = ".+quick.+";
  private static final Pattern QUICK_PATTERN = Pattern.compile("QuIcK",
Pattern.CASE_INSENSITIVE | Pattern.DOTALL);

  Filter basicFilter;
  Filter nullFilter;
  Filter substrFilter;
  Filter regexFilter;
  Filter regexPatternFilter;

  @Before
  public void setUp() throws Exception {
    basicFilter = basicFilterNew();
    nullFilter = nullFilterNew();
    substrFilter = substrFilterNew();
    regexFilter = regexFilterNew();
    regexPatternFilter = regexFilterNew(QUICK_PATTERN);
  }

  private Filter basicFilterNew() {
    return new SingleColumnValueFilter(COLUMN_FAMILY, COLUMN_QUALIFIER,
      CompareOp.GREATER_OR_EQUAL, VAL_2);
  }

  private Filter nullFilterNew() {
    return new SingleColumnValueFilter(COLUMN_FAMILY, COLUMN_QUALIFIER,
CompareOp.NOT_EQUAL,
        new NullComparator());
  }

  private Filter substrFilterNew() {
    return new SingleColumnValueFilter(COLUMN_FAMILY, COLUMN_QUALIFIER,
      CompareOp.EQUAL,
      new SubstringComparator(QUICK_SUBSTR));
  }

  private Filter regexFilterNew() {
    return new SingleColumnValueFilter(COLUMN_FAMILY, COLUMN_QUALIFIER,
      CompareOp.EQUAL,
      new RegexStringComparator(QUICK_REGEX));
  }

  private Filter regexFilterNew(Pattern pattern) {
    return new SingleColumnValueFilter(COLUMN_FAMILY, COLUMN_QUALIFIER,
        CompareOp.EQUAL,
        new RegexStringComparator(pattern.pattern(), pattern.flags()));
  }
```

```
private void basicFilterTests(SingleColumnValueFilter filter)
    throws Exception {
  KeyValue kv = new KeyValue(ROW, COLUMN_FAMILY, COLUMN_QUALIFIER, VAL_2);
  assertTrue("basicFilter1", filter.filterKeyValue(kv) == Filter.ReturnCode.INCLUDE);

  kv = new KeyValue(ROW, COLUMN_FAMILY, COLUMN_QUALIFIER, VAL_3);
  assertTrue("basicFilter2", filter.filterKeyValue(kv) == Filter.ReturnCode.INCLUDE);

  kv = new KeyValue(ROW, COLUMN_FAMILY, COLUMN_QUALIFIER, VAL_4);
  assertTrue("basicFilter3", filter.filterKeyValue(kv) == Filter.ReturnCode.INCLUDE);

  assertFalse("basicFilterNotNull", filter.filterRow());
  filter.reset();
  kv = new KeyValue(ROW, COLUMN_FAMILY, COLUMN_QUALIFIER, VAL_1);
  assertTrue("basicFilter4", filter.filterKeyValue(kv) == Filter.ReturnCode.NEXT_ROW);

  kv = new KeyValue(ROW, COLUMN_FAMILY, COLUMN_QUALIFIER, VAL_2);
  assertTrue("basicFilter4", filter.filterKeyValue(kv) == Filter.ReturnCode.NEXT_ROW);

  assertFalse("basicFilterAllRemaining", filter.filterAllRemaining());
  assertTrue("basicFilterNotNull", filter.filterRow());
  filter.reset();
  filter.setLatestVersionOnly(false);
  kv = new KeyValue(ROW, COLUMN_FAMILY, COLUMN_QUALIFIER, VAL_1);
  assertTrue("basicFilter5", filter.filterKeyValue(kv) == Filter.ReturnCode.INCLUDE);

  kv = new KeyValue(ROW, COLUMN_FAMILY, COLUMN_QUALIFIER, VAL_2);
  assertTrue("basicFilter5", filter.filterKeyValue(kv) == Filter.ReturnCode.INCLUDE);

  assertFalse("basicFilterNotNull", filter.filterRow());
}

private void nullFilterTests(Filter filter) throws Exception {
  ((SingleColumnValueFilter) filter).setFilterIfMissing(true);
  KeyValue kv = new KeyValue(ROW, COLUMN_FAMILY, COLUMN_QUALIFIER, FULLSTRING_1);
  assertTrue("null1", filter.filterKeyValue(kv) == Filter.ReturnCode.INCLUDE);
  assertFalse("null1FilterRow", filter.filterRow());
  filter.reset();
  kv = new KeyValue(ROW, COLUMN_FAMILY, Bytes.toBytes("qual2"), FULLSTRING_2);
  assertTrue("null2", filter.filterKeyValue(kv) == Filter.ReturnCode.INCLUDE);
  assertTrue("null2FilterRow", filter.filterRow());
}

private void substrFilterTests(Filter filter)
    throws Exception {
  KeyValue kv = new KeyValue(ROW, COLUMN_FAMILY, COLUMN_QUALIFIER,
    FULLSTRING_1);
  assertTrue("substrTrue",
    filter.filterKeyValue(kv) == Filter.ReturnCode.INCLUDE);
  kv = new KeyValue(ROW, COLUMN_FAMILY, COLUMN_QUALIFIER,
    FULLSTRING_2);
  assertTrue("substrFalse", filter.filterKeyValue(kv) == Filter.ReturnCode.INCLUDE);
  assertFalse("substrFilterAllRemaining", filter.filterAllRemaining());
  assertFalse("substrFilterNotNull", filter.filterRow());
}

private void regexFilterTests(Filter filter)
    throws Exception {
  KeyValue kv = new KeyValue(ROW, COLUMN_FAMILY, COLUMN_QUALIFIER,
    FULLSTRING_1);
  assertTrue("regexTrue",
    filter.filterKeyValue(kv) == Filter.ReturnCode.INCLUDE);
  kv = new KeyValue(ROW, COLUMN_FAMILY, COLUMN_QUALIFIER,
    FULLSTRING_2);
  assertTrue("regexFalse", filter.filterKeyValue(kv) == Filter.ReturnCode.INCLUDE);
  assertFalse("regexFilterAllRemaining", filter.filterAllRemaining());
  assertFalse("regexFilterNotNull", filter.filterRow());
}

private void regexPatternFilterTests(Filter filter)
    throws Exception {
  KeyValue kv = new KeyValue(ROW, COLUMN_FAMILY, COLUMN_QUALIFIER,
    FULLSTRING_1);
```

```
    assertTrue("regexTrue",
      filter.filterKeyValue(kv) == Filter.ReturnCode.INCLUDE);
    assertFalse("regexFilterAllRemaining", filter.filterAllRemaining());
    assertFalse("regexFilterNotNull", filter.filterRow());
  }

  private Filter serializationTest(Filter filter)
      throws Exception {
    // Decompose filter to bytes.
    byte[] buffer = filter.toByteArray();

    // Recompose filter.
    Filter newFilter = SingleColumnValueFilter.parseFrom(buffer);
    return newFilter;
  }

  /**
   * Tests identification of the stop row
   * @throws Exception
   */
  @Test
  public void testStop() throws Exception {
    basicFilterTests((SingleColumnValueFilter) basicFilter);
    nullFilterTests(nullFilter);
    substrFilterTests(substrFilter);
    regexFilterTests(regexFilter);
    regexPatternFilterTests(regexPatternFilter);
  }

  /**
   * Tests serialization
   * @throws Exception
   */
  @Test
  public void testSerialization() throws Exception {
    Filter newFilter = serializationTest(basicFilter);
    basicFilterTests((SingleColumnValueFilter)newFilter);
    newFilter = serializationTest(nullFilter);
    nullFilterTests(newFilter);
    newFilter = serializationTest(substrFilter);
    substrFilterTests(newFilter);
    newFilter = serializationTest(regexFilter);
    regexFilterTests(newFilter);
    newFilter = serializationTest(regexPatternFilter);
    regexPatternFilterTests(newFilter);
  }

}
```

## Using the HBCK2 Tool to Remediate HBase Clusters

The HBCK2 tool is a repair tool to remediate Apache HBase clusters in CDH. The HBCK2 tool is the next version of the Apache HBase hbck tool.

To identify a list of inconsistencies or blockages in a running HBase cluster, you can view or search the logs using the log search feature in Cloudera Manager. Once you have identified the issue, you can then use the HBCK2 tool to fix the defect or to skip-over a bad state. The HBCK2 tool uses an interactive fix-it process by asking the Master to make the fixes rather than carry out the repair locally.

The HBCK2 performs a single, discrete task each time it is run. The HBCK2 tool does not analyze everything in a running cluster and repair all the problems. Instead, you can use the HBCK2 tool to iteratively find and fix issues in your cluster. The HBCK2 tool lets you use interactive commands to fix one issue at a time.

**Important:** The HBCK2 tool is specific to internals of Apache HBase. Using this tool requires binaries that are specific to your version of CDH, and you must always use it with the assistance of Cloudera Support and/or Cloudera Professional Services. Please contact Cloudera Support if you believe you have an issue that requires using the HBCK2 tool.

## Supported Versions

You can use the HBCK2 tool with these versions of CDH:

- CDH 6.1.x
- CDH 6.2.x
- CDH 6.3.x and later

**Note:** The HBCK2 tool is not supported on CDH 6.0.x. You must upgrade to CDH 6.1.x if you want to use the tool. For information about the supported commands in specific versions of CDH, see HBCK2 Tool Command Reference.

## Running the HBCK2 Tool

The HBCK2 tool is a part of the hbase-operator-tools binary. Once you get the hbase-operator-tools binary from Cloudera, upload the binary tarball to the target cluster and extract the tarball. The HBCK2 JAR file is contained in the operator tools tarball provided by Cloudera Support at `hbase-operator-tools-<version>/hbase-hbck2/hbase-hbck2-<version>.jar`.

You can run the HBCK2 tool by specifying the JAR path with the "-j" option as shown here:

```
$ hbase hbck -j $HOME/hbase-operator-tools-<version>/hbase-hbck2/hbase-hbck2-<version>.jar
```

When you run the command, the HBCK2 tool command-line menu appears.

As a Cloudera Support or Professional Services personnel using this tool to remediate an HBase cluster, gather useful information using these commands as an HBase super user (typically, hbase), or an HBase principal if Kerberos is enabled:

```
$ hdfs dfs -ls -R /hbase/ 2>&1 | tee /tmp/hdfs-ls.txt
$ hbase hbck -details 2>&1 | tee /tmp/hbase-hbck.txt
$ echo "scan 'hbase:meta'" | hbase shell 2>&1 | tee /tmp/hbase-meta.txt
```

## Finding Issues

The HBCK2 tool enables you to use interactive commands to fix one issue at a time. If you have multiple issues, you may have to run the tool iteratively to find and resolve all the issues. Use the following utilities and commands to find the issues.

### Find issues using diagnostic tools

**Master logs**

The Apache HBase Master runs all the cluster start and stop operations, RegionServer assignment, and server crash handling. Everything that the Master does is a procedure on a state machine engine and each procedure has an unique procedure ID (PID). You can trace the lifecycle of a procedure by tracking its PID through the entries in the Master log. Some procedures may spawn sub-procedures and wait for the sub-procedure to complete.

You can trace the sub-procedure by tracking its PID and the parent PID (PPID).

If there is a problem with RegionServer assignment, the Master prints a STUCK log entry similar to the following:

```
2018-09-12 15:29:06,558 WARN
org.apache.hadoop.hbase.master.assignment.AssignmentManager: STUCK
Region-In-Transition rit=OPENING, location=va1001.example.org,00001,1000173230599,
table=IntegrationTestBigLinkedList_20180626110336,
region=dbdb56242f17610c46ea044f7a42895b
```

**Master user interface**

**Status tables**

You can find issues in your HBase tables by looking at the status tables section in the Master user interface home page. Look through the list of tables to identify if a table is ENABLED, ENABLING, DISABLED, or DISABLING. You can also take a look at the regions in transition states: OPEN, CLOSED. For example, there may be an issue if a table is ENABLED, some regions are not in the OPEN state, and the Master log entries do not have any ongoing assignments.

**Procedures and locks**

When an Apache HBase cluster is started, the Procedures & Locks page in the Master user interface is populated with information about the procedures, locks, and the count of WAL files. After the cluster settles, if the WAL file count does not reduce, it leads to procedure blocks. You can identify those procedures and locks on this page.

You can also get a list of locks and procedures using this command in the HBase shell:

```
$ echo "list_locks"| hbase shell &> /tmp/locks.txt
$ echo "list_procedures"| hbase shell &> /tmp/procedures.txt
```

**Apache HBase canary tool**

Use the HBase canary tool to verify the state of the assigns in your cluster. You can run this tool to focus on just one table or the entire cluster. You can check the cluster assign using this command:

```
$ hbase canary -f false -t 6000000 &>/tmp/canary.log
```

Use the -f parameter to look for failed region fetches, and set the -t parameter to run for a specified time.

## Fixing Issues

You must keep these in mind when fixing issues using HBCK2. Ensure that:

- A region is not in the CLOSING state during “assign”, and in the OPENING state during “unassign”. You can change the state using the setRegionState command. See the HBCK2 tool Command Reference section for more information.
- You fix only one table at a time.

**Important:** Contact Cloudera Support before using any of the HBCK2 tool commands.

### Fix assign and unassign issues

You can fix assign and unassign issues by monitoring the current list of outstanding locks. An assign against a locked region will wait till the lock is released. An assignment gets an exclusive lock on the region.

### Fix master startup cannot progress error

If you see a `master startup cannot progress holding-pattern until region online` error in the Master log, it means that the Master is unable to start because there is no procedure to assign hbase:meta. You will see an error message similar to this:

```
2020-04-01 22:07:42,792 WARN org.apache.hadoop.hbase.master.HMaster:
 hbase:meta,,1.1588230740 is NOT online; state={1588230740 state=CLOSING,
```

```
ts=1538456302300, server=ve1017.example.org,22101,1234567891012};
ServerCrashProcedures=true. Master startup cannot progress in holding-pattern until
region onlined.
```

To fix this issue, run the following command:

```
$ hbase hbck -j $HOME/hbase-operator-tools-<version>/hbase-hbck2/hbase-hbck2-<version>.jar

assigns 1588230740
```

The same issue can occur with a hbase:namespace system table. To fix this issue, run the following command:

```
$ hbase hbck -j $HOME/hbase-operator-tools-<version>/hbase-hbck2/hbase-hbck2-<version>.jar

assigns <hbase:namespace encoded region id>
```

You can find the namespace encoded region id using this command:

```
$ echo "scan 'hbase:meta',{COLUMNS=>'info:regioninfo',
FILTER=>\"PrefixFilter('hbase:namespace')\"}" | hbase shell
```

The namespace encoded region id is the value under the "ENCODED" field in the results.

### Fix missing regions in hbase:meta region/table

If you encounter an issue where table regions have been removed from the hbase:meta table, you can use the addFsRegionsMissingInMeta to resolve this issue. Ensure that the Master is online. This command is not as disruptive as the hbase:meta rebuild command.

To fix this issue, run this command:

```
$ hbase hbck -j $HOME/hbase-operator-tools-<version>/hbase-hbck2/hbase-hbck2-<version>.jar

addFsRegionsMissingInMeta <NAMESPACE|NAMESPACE:TABLENAME>
```

The command returns an HBCK2 “assigns” command with all the listed re-inserted regions.You must restart the Master, and then run the HBCK2 'assigns' command returned by the addFsRegionsMissingInMeta command to complete your fix.

Example output:

```
Regions re-added into Meta: 2
WARNING:
2 regions were added to META, but these are not yet on Masters cache.
You need to restart Masters, then run hbck2 'assigns' command below:
assigns 7be03127c5e0e2acfc7cae7ddfa9e29e e50b8c1adc38c942e226a8b2976f0c8c
```

### Fix extra regions in hbase:meta region/table

If there are extra regions in hbase:meta, it may be because of problems in splitting, deleting/moving the region directory manually, or in rare cases because of the loss of metadata.

To fix this issue, run this command:

```
$ hbase hbck -j $HOME/hbase-operator-tools-<version>/hbase-hbck2/hbase-hbck2-<version>.jar

extraRegionsInMeta --fix  <NAMESPACE|NAMESPACE:TABLENAME>...
```

**Important:** Use the --fix option only when the extra regions are overlapping the existing valid regions. Otherwise, use the assigns command to recreate the regions.

### Rebuild hbase:meta

If hbase:meta is offline because it is corrupted, you can bring it back online if the corruption is not too critical. If the namespace region is among the mission regions, scan hbase:meta during initialization to check if hbase:meta is online.

To check if hbase:meta is online, run this command in the Apache HBase shell:

```
$ echo "scan 'hbase:meta', {COLUMN=>'info:regioninfo'}" | hbase shell
```

If this scan does not throw any errors, then you can run the following command to validate that the tables are present:

```
$ hbase hbck -j $HOME/hbase-operator-tools-<version>/hbase-hbck2/hbase-hbck2-<version>.jar
addFsRegionsMissingInMeta <NAMESPACE|NAMESPACE:TABLENAME>
```

The `addFsRegionsMissingInMeta` command adds regions back to the hbase:meta table if the regioninfo file is present in the storage but the regions were deleted because of an issue.

### Fix dropped references and corrupted HFiles

To fix hanging references and corrupt HFiles, run the following command:

```
$ hbase hbck -j $HOME/hbase-operator-tools-<version>/hbase-hbck2/hbase-hbck2-<version>.jar
filesystem --fix [<TABLENAME>...]
```

## HBCK2 Tool Command Reference

```
addFsRegionsMissingInMeta <NAMESPACE|NAMESPACE:TABLENAME>...
```

Options: -d,--force_disable Use this option to abort fix for table if disable fails.

Supported from CDH 6.1.0 and later.

```
assigns [OPTIONS] <ENCODED_REGIONNAME>...
```

Options:-o,--override Use this option to override ownership by another procedure.

Supported in CDH 6.1.0 and later.

```
bypass [OPTIONS] <PID>...
```

Options: -o,--override Use this option to override if procedure is running/stuck -r,--recursive Use this option to bypass parent and its children.

-w,--lockWait Use this option to wait (in milliseconds) before giving up; default=1.

Supported in CDH 6.1.0 and later.

```
extraRegionsInMeta <NAMESPACE|NAMESPACE:TABLENAME>...
```

Options:-f, --fix Use this option to fix meta by removing all extra regions found.

Supported from CDH 6.1.0 and later.

```
filesystem [OPTIONS] [<TABLENAME>...]
```

Options:-f, --fix Use this option to sideline corrupt HFiles, bad links, and references.

Supported in CDH 6.1.0 and later.

```
replication [OPTIONS] [<TABLENAME>...]
```

Options:-f, --fix Use this option to fix replication issues.

Supported in CDH 6.1.0 and later.

```
reportMissingRegionsInMeta <NAMESPACE|NAMESPACE:TABLENAME>...
```

Use this command when regions missing from hbase:meta but directories are still present in HDFS.

Supported in CDH 6.1.0 and later.

```
setRegionState <ENCODED_REGIONNAME> <STATE>
```

Possible region states: OFFLINE, OPENING, OPEN, CLOSING, CLOSED, SPLITTING, SPLIT, FAILED_OPEN, FAILED_CLOSE, MERGING, MERGED, SPLITTING_NEW, MERGING_NEW, ABNORMALLY_CLOSED.

Supported in CDH 6.1.0 and later.

```
setTableState <TABLENAME> <STATE>
```

Possible table states and representations in hbase:meta table: ENABLED (\x08\x00), DISABLED (\x08\x01), DISABLING (\x08\x02), ENABLING (\x08\x03).

Supported in CDH 6.1.0 and later.

```
```

```
scheduleRecoveries <SERVERNAME>...
```

Schedule ServerCrashProcedure(SCP) for list of RegionServers. Format server name as '<HOSTNAME>,<PORT>,<STARTCODE>' .

Supported in CDH 6.2.0 and later.

```
unassigns <ENCODED_REGIONNAME>...
```

Options:-o,--override Use this option to override ownership by another procedure.

Supported in CDH 6.1.0 and later.

## Exposing HBase Metrics to a Ganglia Server

Ganglia is a popular open-source monitoring framework. You can expose HBase metrics to a Ganglia instance so that Ganglia can detect potential problems with your HBase cluster.

### Expose HBase Metrics to Ganglia Using Cloudera Manager

**Minimum Required Role: Cluster Administrator** (also provided by **Full Administrator**)

1. Go to the HBase service.
2. Click the **Configuration** tab.
3. Select the HBase Master or RegionServer role. To monitor both, configure each role as described in the rest of the procedure.
4. Select **Category** > **Metrics**.
5. Locate the **Hadoop Metrics2 Advanced Configuration Snippet (Safety Valve)** property or search for it by typing its name in the Search box.
6. Edit the property. Add the following, substituting the server information with your own.

```
hbase.sink.ganglia.class=org.apache.hadoop.metrics2.sink.ganglia.GangliaSink31
hbase.sink.ganglia.servers=<Ganglia server>:<port>
hbase.sink.ganglia.period=10
```

To apply this configuration property to other role groups as needed, edit the value for the appropriate role group. See Modifying Configuration Properties Using Cloudera Manager.

7. Enter a **Reason for change**, and then click **Save Changes** to commit the changes.
8. Restart the role.
9. Restart the service.

# Managing HBase Security

This topic pulls together content also found elsewhere which relates to configuring and using HBase in a secure environment. For the most part, securing an HBase cluster is a one-way operation, and moving from a secure to an unsecure configuration should not be attempted without contacting Cloudera support for guidance.

## HBase Authentication

To configure HBase security, complete the following tasks:

1. **Configure HBase Authentication**: You must establish a mechanism for HBase servers and clients to securely identify themselves with HDFS, ZooKeeper, and each other. This ensures that hosts are who they claim to be.

   **Note:**

   - To enable HBase to work with Kerberos security, you must perform the installation and configuration steps in Enabling Kerberos Authentication for CDH and ZooKeeper Security Configuration.
   - Although an HBase Thrift server can connect to a secured Hadoop cluster, access is not secured from clients to the HBase Thrift server. To encrypt communication between clients and the HBase Thrift Server, see Configuring TLS/SSL for HBase Thrift Server.

   The following sections describe how to use Apache HBase and CDH 6 with Kerberos security:

   - Configuring Kerberos Authentication for HBase
   - Configuring Secure HBase Replication
   - Configuring the HBase Client TGT Renewal Period

2. **Configure HBase Authorization**: You must establish rules for the resources that clients are allowed to access. For more information, see Configuring HBase Authorization.

### Using the Hue HBase App

Hue includes an HBase App that allows you to interact with HBase through a Thrift proxy server. Because Hue sits between the Thrift server and the client, the Thrift server assumes that all HBase operations come from the hue user and not the client. To ensure that users in Hue are only allowed to perform HBase operations assigned to their own credentials, and not those of the hue user, you must enable HBase impersonation. For more information about the how to enable doAs Impersonation for the HBase Browser Application, see Enabling the HBase Browser Application with doAs Impersonation.

## Configuring HBase Authorization

After configuring HBase authentication (as detailed in HBase Configuration), you must define rules on resources that is allowed to access. HBase rules can be defined for individual tables, columns, and cells within a table. Cell-level authorization is fully supported since CDH 5.2.

**Important:** In a cluster managed by Cloudera Manager, HBase authorization is disabled by default. You have to enable HBase authorization (as detailed in Enable HBase Authorization) to use any kind of authorization method.

## Understanding HBase Access Levels

HBase access levels are granted independently of each other and allow for different types of operations at a given scope.

- **Read (R)** - can read data at the given scope
- **Write (W)** - can write data at the given scope
- **Execute (X)** - can execute coprocessor endpoints at the given scope
- **Create (C)** - can create tables or drop tables (even those they did not create) at the given scope
- **Admin (A)** - can perform cluster operations such as balancing the cluster or assigning regions at the given scope

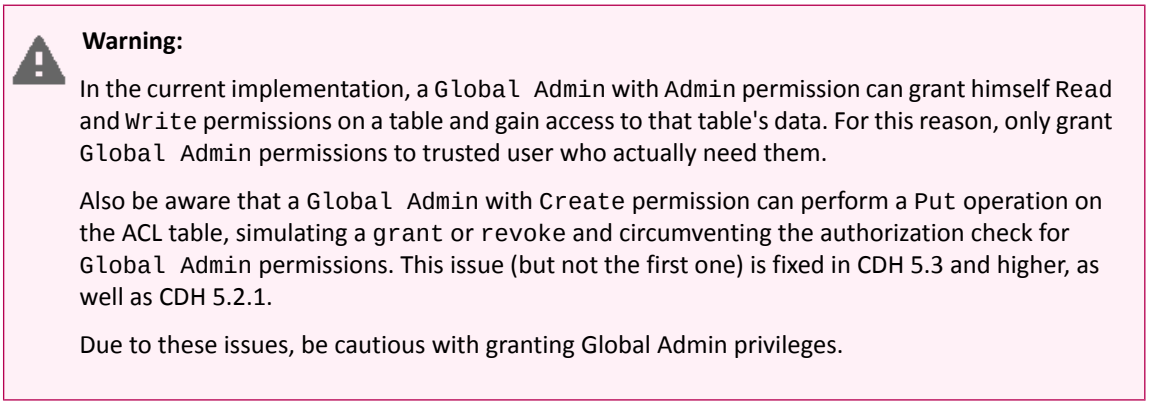
The possible scopes are:

- **Superuser** - superusers can perform any operation available in HBase, to any resource. The user who runs HBase on your cluster is a superuser, as are any principals assigned to the configuration property `hbase.superuser` in `hbase-site.xml` on the HMaster.
- **Global** - permissions granted at `global` scope allow the admin to operate on all tables of the cluster.
- **Namespace** - permissions granted at `namespace` scope apply to all tables within a given namespace.
- **Table** - permissions granted at `table` scope apply to data or metadata within a given table.
- **ColumnFamily** - permissions granted at `ColumnFamily` scope apply to cells within that ColumnFamily.
- **Cell** - permissions granted at Cell scope apply to that exact cell coordinate. This allows for policy evolution along with data. To change an ACL on a specific cell, write an updated cell with new ACL to the precise coordinates of the original. If you have a multi-versioned schema and want to update ACLs on all visible versions, you'll need to write new cells for all visible versions. The application has complete control over policy evolution. The exception is `append` and `increment` processing. `Appends` and `increments` can carry an ACL in the operation. If one is included in the operation, then it will be applied to the result of the `append` or `increment`. Otherwise, the ACL of the existing cell being appended to or incremented is preserved.

The combination of access levels and scopes creates a matrix of possible access levels that can be granted to a user. In a production environment, it is useful to think of access levels in terms of what is needed to do a specific job. The following list describes appropriate access levels for some common types of HBase users. It is important not to grant more access than is required for a given user to perform their required tasks.

- **Superusers** - In a production system, only the HBase user should have superuser access. In a development environment, an administrator might need superuser access to quickly control and manage the cluster. However, this type of administrator should usually be a `Global Admin` rather than a superuser.
- **Global Admins** - A `global admin` can perform tasks and access every table in HBase. In a typical production environment, an admin should not have `Read` or `Write` permissions to data within tables.
  - A global admin with `Admin` permissions can perform cluster-wide operations on the cluster, such as balancing, assigning or unassigning regions, or calling an explicit major compaction. This is an operations role.
  - A global admin with `Create` permissions can create or drop any table within HBase. This is more of a DBA-type role.

  In a production environment, it is likely that different users will have only one of `Admin` and `Create` permissions.



**Warning:**

In the current implementation, a `Global Admin` with `Admin` permission can grant himself `Read` and `Write` permissions on a table and gain access to that table's data. For this reason, only grant `Global Admin` permissions to trusted user who actually need them.

Also be aware that a `Global Admin` with `Create` permission can perform a `Put` operation on the ACL table, simulating a `grant` or `revoke` and circumventing the authorization check for `Global Admin` permissions. This issue (but not the first one) is fixed in CDH 5.3 and higher, as well as CDH 5.2.1.

Due to these issues, be cautious with granting Global Admin privileges.

- **Namespace Admin** - a namespace admin with Create permissions can create or drop tables within that namespace, and take and restore snapshots. A namespace admin with Admin permissions can perform operations such as splits or major compactions on tables within that namespace. Prior to CDH 5.4, only global admins could create namespaces. In CDH 5.4, any user with Namespace Create privileges can create namespaces.
- **Table Admins** - A table admin can perform administrative operations only on that table. A table admin with `Create` permissions can create snapshots from that table or restore that table from a snapshot. A table admin with `Admin` permissions can perform operations such as splits or major compactions on that table.
- **Users** - Users can read or write data, or both. Users can also execute coprocessor endpoints, if given `Executable` permissions.

**Important:**

If you are using Kerberos principal names when setting ACLs for users, Hadoop uses only the first part (short) of the Kerberos principal when converting it to the username. Hence, for the principal `ann/fully.qualified.domain.name@YOUR-REALM.COM`, HBase ACLs should only be set for user `ann`.

The following table shows some typical job descriptions at a hypothetical company and the permissions they might require to get their jobs done using HBase.

**Table 14: Real-World Example of Access Levels**

| Job Title | Scope | Permissions | Description |
|---|---|---|---|
| Senior Administrator | Global | Admin, Create | Manages the cluster and gives access to Junior Administrators. |
| Junior Administrator | Global | Create | Creates tables and gives access to Table Administrators. |
| Table Administrator | Table | Admin | Maintains a table from an operations point of view. |
| Data Analyst | Table | Read | Creates reports from HBase data. |
| Web Application | Table | Read, Write | Puts data into HBase and uses HBase data to perform operations. |

### Further Reading

- Access Control Matrix
- Security - Apache HBase Reference Guide

## Enable HBase Authorization

HBase authorization is built on top of the Coprocessors framework, specifically `AccessController Coprocessor`.

**Note:** Once the Access Controller coprocessor is enabled, any user who uses the HBase shell will be subject to access control. Access control will also be in effect for native (Java API) client access to HBase.

1. Go to **Clusters** and select the HBase cluster.
2. Select **Configuration**.

3. Search for **HBase Secure Authorization** and select it.
4. Search for **HBase Service Advanced Configuration Snippet (Safety Valve) for hbase-site.xml** and enter the following into it to enable `hbase.security.exec.permission.checks`. Without this option, all users will continue to have access to execute endpoint coprocessors. This option is not enabled when you enable HBase Secure Authorization for backward compatibility.

```
<property>
  <name>hbase.security.exec.permission.checks</name>
  <value>true</value>
</property>
```

5. Optionally, search for and configure **HBase Coprocessor Master Classes** and **HBase Coprocessor Region Classes**.

## Configure Access Control Lists for Authorization

Now that HBase has the security coprocessor enabled, you can set ACLs using the HBase shell. Start the HBase shell as usual.

**Important:**

The host running the shell must be configured with a keytab file as described in Configuring Kerberos Authentication for HBase.

The commands that control ACLs take the following form. Group names are prefixed with the @ symbol.

```
hbase> grant <user> <permissions> [ @<namespace> [ <table>[ <column family>[ <column
qualifier> ] ] ] ]   # grants permissions

hbase> revoke <user> [ @<namespace> [ <table> [ <column family> [ <column qualifier> ]
 ] ]  # revokes permissions

hbase> user_permission <table>
                    # displays existing permissions
```

In the above commands, fields encased in <> are variables, and fields in `[]` are optional. The `permissions` variable must consist of zero or more character from the set "RWCA".

- `R` denotes read permissions, which is required to perform `Get`, `Scan`, or `Exists` calls in a given scope.
- `W` denotes write permissions, which is required to perform `Put`, `Delete`, `LockRow`, `UnlockRow`, `IncrementColumnValue`, `CheckAndDelete`, `CheckAndPut`, `Flush`, or `Compact` in a given scope.
- `X` denotes execute permissions, which is required to execute coprocessor endpoints.
- `C` denotes create permissions, which is required to perform `Create`, `Alter`, or `Drop` in a given scope.
- `A` denotes admin permissions, which is required to perform `Enable`, `Disable`, `Snapshot`, `Restore`, `Clone`, `Split`, `MajorCompact`, `Grant`, `Revoke`, and `Shutdown` in a given scope.

**Access Control List Example Commands**

```
grant 'user1', 'RWC'
grant 'user2', 'RW', 'tableA'
grant 'user3', 'C', '@my_namespace'
```

Be sure to review the information in Understanding HBase Access Levels to understand the implications of the different access levels.

### Configure Cell_Level Access Control Lists

If you wish to enable cell-level ACLs for HBase, then you must modify the default values for the following properties:

```
hbase.security.exec.permission.checks => true (the default value is false)
hbase.security.access.early_out => false (the default value is true)
hfile.format.version => 3 (the default value is 2)
```

Unless you modify the default properties as specified (or via the service-wide **HBase Service Advanced Configuration Snippet (Safety Valve) for hbase-site.xml**, which requires a service restart), then cell level ACLs will not work.

The following example shows how to grant (or revoke) HBase permissions (in this case, `read` permission) at the cell-level via an ACL:

```
grant 'Employee', { 'employe.name' => 'R' }, { COLUMNS => [ 'pd' ], FILTER =>
"(PrefixFilter ('T'))" }
```

## Auditing HBase Authorization Grants

When Cloudera Navigator collects HBase audits (enabled by default in Cloudera Manager), each time a grant occurs, Navigator collects an audit event for the grant. The audit includes the information in the following table:

| Field | Description |
|---|---|
| ID | Automatically generated and incremented ID. |
| SERVICE_NAME | HBase service name as shown in Cloudera Manager. |
| ALLOWED | Whether the request to perform an operation failed or succeeded. A failure occurs if the user is not authorized to perform the action. |
| USERNAME | Username for the account that performed the operation. Depending on how HBase is configured, this can be the service user. |
| IMPERSONATOR | If HBase is configured for the service user to impersonate the user performing an operation, the impersonated user appears in USERNAME and the service name appears in this field. |
| IP_ADDR | IP address of the HBase service. |
| EVENT_TIME | Event occurrence in milliseconds in epoch time format. |
| TABLE_NAME | The object that is the target of the operation, if appropriate. |
| FAMILY | The column family that is the object of the operation, if appropriate. |
| QUALIFIER | The column that is the object of the operation, if appropriate. |
| OPERATION | The operation command. |

For information on other HBase commands audited through Cloudera Navigator, see Cloudera Navigator Service Audit Events.

# Configuring the HBase Thrift Server Role

**Minimum Required Role:** Cluster Administrator (also provided by **Full Administrator**)

The Thrift Server role is not added by default when you install HBase, but it is required before you can use certain other features such as the Hue HBase browser. To add the Thrift Server role:

1. Go to the HBase service.
2. Click the **Instances** tab.
3. Click the **Add Role Instances** button.

4. Select the host(s) where you want to add the Thrift Server role (you only need one for Hue) and click **Continue**. The Thrift Server role should appear in the instances list for the HBase server.
5. Select the Thrift Server role instance.
6. Select **Actions for Selected** > **Start**.

## Other HBase Security Topics

- Using BulkLoad On A Secure Cluster on page 65
- Configuring Secure HBase Replication

# Troubleshooting HBase

The Cloudera HBase packages have been configured to place logs in `/var/log/hbase`. Cloudera recommends tailing the `.log` files in this directory when you start HBase to check for any error messages or failures.

## Table Creation Fails after Installing LZO

If you install LZO after starting the RegionServer, you will not be able to create a table with LZO compression until you re-start the RegionServer.

**Why this happens**

When the RegionServer starts, it runs CompressionTest and caches the results. When you try to create a table with a given form of compression, it refers to those results. You have installed LZO since starting the RegionServer, so the cached results, which pre-date LZO, cause the create to fail.

**What to do**

Restart the RegionServer. Now table creation with LZO will succeed.

## Thrift Server Crashes after Receiving Invalid Data

The Thrift server may crash if it receives a large amount of invalid data, due to a buffer overrun.

**Why this happens**

The Thrift server allocates memory to check the validity of data it receives. If it receives a large amount of invalid data, it may need to allocate more memory than is available. This is due to a limitation in the Thrift library itself.

**What to do**

To prevent the possibility of crashes due to buffer overruns, use the framed and compact transport protocols. These protocols are disabled by default, because they may require changes to your client code. The two options to add to your hbase-site.xml are `hbase.regionserver.thrift.framed` and `hbase.regionserver.thrift.compact`. Set each of these to `true`, as in the XML below. You can also specify the maximum frame size, using the `hbase.regionserver.thrift.framed.max_frame_size_in_mb` option.

```
<property>
  <name>hbase.regionserver.thrift.framed</name>
  <value>true</value>
</property>
<property>
  <name>hbase.regionserver.thrift.framed.max_frame_size_in_mb</name>
  <value>2</value>
</property>
<property>
  <name>hbase.regionserver.thrift.compact</name>
  <value>true</value>
</property>
```

## HBase is using more disk space than expected.

HBase StoreFiles (also called HFiles) store HBase row data on disk. HBase stores other information on disk, such as write-ahead logs (WALs), snapshots, data that would otherwise be deleted but would be needed to restore from a stored snapshot.

**Warning:** The following information is provided to help you troubleshoot high disk usage only. Do not edit or remove any of this data outside the scope of the HBase APIs or HBase Shell, or your data is very likely to become corrupted.

**Table 15: HBase Disk Usage**

| Location | Purpose | Troubleshooting Notes |
|---|---|---|
| `/hbase/.snapshots` | Contains one subdirectory per snapshot. | To list snapshots, use the HBase Shell command `list_snapshots`. To remove a snapshot, use `delete_snapshot`. |
| `/hbase/.archive` | Contains data that would otherwise have been deleted (either because it was explicitly deleted or expired due to TTL or version limits on the table) but that is required to restore from an existing snapshot. | To free up space being taken up by excessive archives, delete the snapshots that refer to them. Snapshots never expire so data referred to by them is kept until the snapshot is removed. Do not remove anything from `/hbase/.archive` manually, or you will corrupt your snapshots. |
| `/hbase/.logs` | Contains HBase WAL files that are required to recover regions in the event of a RegionServer failure. | WALs are removed when their contents are verified to have been written to StoreFiles. Do not remove them manually. If the size of any subdirectory of `/hbase/.logs/` is growing, examine the HBase server logs to find the root cause for why WALs are not being processed correctly. |
| `/hbase/logs/.oldWALs` | Contains HBase WAL files that have already been written to disk. A HBase maintenance thread removes them periodically based on a TTL. | To tune the length of time a WAL stays in the `.oldWALs` before it is removed, configure the `hbase.master.logcleaner.ttl` property, which defaults to 60000 milliseconds, or 1 hour. |
| `/hbase/.logs/.corrupt` | Contains corrupted HBase WAL files. | Do not remove corrupt WALs manually. If the size of any subdirectory of `/hbase/.logs/` is growing, examine the HBase server logs to find the root cause for why WALs are not being processed correctly. |

# Appendix: Apache License, Version 2.0

**SPDX short identifier: Apache-2.0**

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims

licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution.

You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

1. You must give any other recipients of the Work or Derivative Works a copy of this License; and
2. You must cause any modified files to carry prominent notices stating that You changed the files; and
3. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
4. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions.

Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks.

This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty.

Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability.

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability.

While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

### APPENDIX: How to apply the Apache License to your work

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

```
Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

  http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```