cloudera®

Apache HBase Guide

Important Notice
© 2010-2020 Cloudera, Inc. All rights reserved.

Cloudera, the Cloudera logo, and any other product or
service names or slogans contained in this document are trademarks of Cloudera and its suppliers or licensors, and may not be copied, imitated or used, in whole or in part, without the prior written permission of Cloudera or the applicable trademark holder. If this documentation includes code, including but not limited to, code examples, Cloudera makes this available to you under the terms of the Apache License, Version 2.0, including any required notices. A copy of the Apache License Version 2.0, including any notices, is included herein. A copy of the Apache License Version 2.0 can also be found here: https://opensource.org/licenses/Apache-2.0

Hadoop and the Hadoop elephant logo are trademarks of the Apache Software Foundation. All other trademarks, registered trademarks, product names and company names or logos mentioned in this document are the property of their respective owners. Reference to any products, services, processes or other information, by trade name, trademark, manufacturer, supplier or otherwise does not constitute or imply endorsement, sponsorship or recommendation thereof by us.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Cloudera.

Cloudera may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Cloudera, the furnishing of this document does not give you any license to these patents, trademarks copyrights, or other intellectual property. For information about patents covering Cloudera products, see http://tiny.cloudera.com/patents.

The information in this document is subject to change without notice. Cloudera shall not be liable for any damages resulting from technical errors or omissions which may be present in this document, or from use of this document.

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 5.14.x
Date: May 21, 2020

Table of Contents

Apache HBase Guide.....9

Configuration Settings.....9

Managing HBase.....9

HBase Security.....9

HBase Replication.....9

HBase High Availability.....10

Troubleshooting HBase.....10

Upstream Information for HBase.....10

HBase Installation.....11

New Features and Changes for HBase in CDH 5.....11

CDH 5.4 HBase Changes.....11

CDH 5.3 HBase Changes.....13

`SlabCache` Has Been Deprecated.....13

`checkAndMutate(RowMutations)` API.....13

CDH 5.2 HBase Changes.....13

CDH 5.1 HBase Changes.....16

CDH 5.0.x HBase Changes.....20

Installing HBase.....21

Starting HBase in Standalone Mode.....22

Installing the HBase Master.....22

Starting the HBase Master.....22

Installing and Starting the HBase Thrift Server.....23

Installing and Configuring HBase REST.....23

Configuring HBase in Pseudo-Distributed Mode.....24

Modifying the HBase Configuration.....24

Creating the /hbase Directory in HDFS.....24

Enabling Servers for Pseudo-distributed Operation.....25

Installing and Starting the HBase Thrift Server.....26

Deploying HBase on a Cluster.....26

Choosing Where to Deploy the Processes.....26

Configuring for Distributed Operation.....27

Accessing HBase by using the HBase Shell.....27

HBase Shell Overview.....28

Setting Virtual Machine Options for HBase Shell.....28

Scripting with HBase Shell.....28

Configuring HBase Online Merge....28
Using MapReduce with HBase....29
Troubleshooting HBase....29
Table Creation Fails after Installing LZO....29
Thrift Server Crashes after Receiving Invalid Data....30
HBase is using more disk space than expected....30

Upgrading HBase....32
Upgrading HBase from a Lower CDH 5 Release....32

Configuration Settings for HBase....34
Using DNS with HBase....34
Using the Network Time Protocol (NTP) with HBase....34
Setting User Limits for HBase....34
Using `dfs.datanode.max.transfer.threads` with HBase....36
Configuring BucketCache in HBase....36
Configuring Encryption in HBase....36
Configuring Cell Level TTL in HBase....36
Using Hedged Reads....37
Accessing HBase by using the HBase Shell....37
HBase Shell Overview....37
Setting Virtual Machine Options for HBase Shell....38
Scripting with HBase Shell....38
Configuring HBase Online Merge....38
Configuring RegionServer Grouping....39
Configuring the BlockCache....42
Configuring the Scanner Heartbeat....42
Troubleshooting HBase....42

Managing HBase....43
Creating the HBase Root Directory....43
Graceful Shutdown....43
Configuring the HBase Thrift Server Role....44
Enabling HBase Indexing....44
Adding a Custom Coprocessor....44
Disabling Loading of Coprocessors....45
Enabling Hedged Reads on HBase....45
Advanced Configuration for Write-Heavy Workloads....45
Managing HBase....46
Creating the HBase Root Directory....46

Graceful Shutdown....46
Configuring the HBase Thrift Server Role....47
Enabling HBase Indexing....47
Adding a Custom Coprocessor....47
Disabling Loading of Coprocessors....48
Enabling Hedged Reads on HBase....48
Advanced Configuration for Write-Heavy Workloads....48
Starting and Stopping HBase....49
Starting or Restarting HBase....49
Stopping HBase....49
Starting and Stopping HBase Using the Command Line....49
Accessing HBase by using the HBase Shell....50
HBase Shell Overview....50
Setting Virtual Machine Options for HBase Shell....51
Scripting with HBase Shell....51
Using HBase Command-Line Utilities....51
`PerformanceEvaluation`....51
`LoadTestTool`....52
`wal`....53
`hfile`....54
`hbck`....54
`clean`....55
Configuring HBase Garbage Collection....55
Configure HBase Garbage Collection Using Cloudera Manager....56
Configure HBase Garbage Collection Using the Command Line....56
Disabling the `BoundedByteBufferPool`....56
Configuring the HBase Canary....57
Configure the HBase Canary Using Cloudera Manager....57
Configure the HBase Canary Using the Command Line....58
Checking and Repairing HBase Tables....58
Running `hbck` Manually....58
Hedged Reads....59
Enabling Hedged Reads for HBase Using Cloudera Manager....59
Enabling Hedged Reads for HBase Using the Command Line....60
Monitoring the Performance of Hedged Reads....60
Configuring the Blocksize for HBase....60
Configuring the Blocksize for a Column Family....61
Monitoring Blocksize Metrics....61
Configuring the HBase BlockCache....61
Contents of the BlockCache....61
Deciding Whether To Use the BucketCache....62
Bypassing the BlockCache....62
Cache Eviction Priorities....62
Sizing the BlockCache....63

About the Off-heap BucketCache.....63
Configuring the Off-heap BucketCache.....63
Configuring the HBase Scanner Heartbeat.....68
Configure the Scanner Heartbeat Using Cloudera Manager.....68
Configure the Scanner Heartbeat Using the Command Line.....68
Limiting the Speed of Compactions.....69
Configure the Compaction Speed Using Cloudera Manager.....69
Configure the Compaction Speed Using the Command Line.....69
Reading Data from HBase.....70
Hedged Reads.....72
Enabling Hedged Reads for HBase Using the Command Line.....72
HBase Filtering.....72
Writing Data to HBase.....80
Importing Data Into HBase.....82
Choosing the Right Import Method.....82
Using CopyTable.....82
Using Snapshots.....83
Using BulkLoad.....84
Using Cluster Replication.....86
Using Pig and HCatalog.....89
Using the Java API.....90
Using the Apache Thrift Proxy API.....90
Using the REST Proxy API.....92
Using Flume.....92
Using Spark.....94
Using Spark and Kafka.....95
Using a Custom MapReduce Job.....96
Configuring and Using the HBase REST API.....96
Installing the REST Server.....97
Using the REST API.....97
Configuring HBase MultiWAL Support.....105
Configuring MultiWAL Support Using Cloudera Manager.....105
Configuring MultiWAL Support Using the Command Line.....105
Storing Medium Objects (MOBs) in HBase.....106
Configuring Columns to Store MOBs.....106
HBase MOB Cache Properties.....107
Configuring the MOB Cache Using Cloudera Manager.....107
Configuring the MOB Cache Using the Command Line.....108
Testing MOB Storage and Retrieval Performance.....108
Compacting MOB Files Manually.....108
Configuring the Storage Policy for the Write-Ahead Log (WAL).....109
Exposing HBase Metrics to a Ganglia Server.....110
Expose HBase Metrics to Ganglia Using Cloudera Manager.....110
Expose HBase Metrics to Ganglia Using the Command Line.....110

Managing HBase Security.....111
HBase Authentication.....111
Configuring HBase Authorization.....111
Understanding HBase Access Levels.....112
Enable HBase Authorization.....113
Configure Access Control Lists for Authorization.....114
Configuring the HBase Thrift Server Role.....115
Other HBase Security Topics.....116

HBase Replication.....117
Common Replication Topologies.....117
Notes about Replication.....118
Requirements.....118
Deploying HBase Replication.....118
Configuring Secure Replication.....120
Disabling Replication at the Peer Level.....122
Stopping Replication in an Emergency.....122
Creating the Empty Table On the Destination Cluster.....123
Initiating Replication When Data Already Exists.....123
Understanding How WAL Rolling Affects Replication.....124
Configuring Secure HBase Replication.....125
Restoring Data From A Replica.....125
Verifying that Replication is Working.....125
Replication Caveats.....127

HBase High Availability.....128
Enabling HBase High Availability Using Cloudera Manager.....128
Enabling HBase High Availability Using the Command Line.....128
HBase Read Replicas.....128
Timeline Consistency.....129
Keeping Replicas Current.....129
Enabling Read Replica Support.....129
Configuring Rack Awareness for Read Replicas.....132
Activating Read Replicas On a Table.....132
Requesting a Timeline-Consistent Read.....132

Troubleshooting HBase.....134
Table Creation Fails after Installing LZO.....134
Thrift Server Crashes after Receiving Invalid Data.....134

HBase is using more disk space than expected....134
HiveServer2 Security Configuration....135
Enabling Kerberos Authentication for HiveServer2....136
Using LDAP Username/Password Authentication with HiveServer2....137
Configuring LDAPS Authentication with HiveServer2....138
Pluggable Authentication....139
Trusted Delegation with HiveServer2....139
HiveServer2 Impersonation....140
Securing the Hive Metastore....141
Disabling the Hive Security Configuration....141
Hive Metastore Server Security Configuration....142
Using Hive to Run Queries on a Secure HBase Server....143

Appendix: Apache License, Version 2.0....144

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 34.

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. For more information, see Configuring HBase in Pseudo-Distributed Mode on page 24.

Managing HBase

You can manage and configure various aspects of HBase using Cloudera Manager. For more information, see Managing HBase on page 43.

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 111.

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 5, 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 on page 117.

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 on page 128.

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 29.

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.

HBase Installation

Apache HBase provides large-scale tabular storage for Hadoop using the Hadoop Distributed File System (HDFS). Cloudera recommends installing HBase in a standalone mode before you try to run it on a whole cluster.

Note: Install Cloudera Repository

Before using the instructions on this page to install or upgrade:

- Install the Cloudera `yum`, `zypper/YaST` or `apt` repository.
- Install or upgrade CDH 5 and make sure it is functioning correctly.

For instructions, see Installing the Latest CDH 5 Release and Upgrading Unmanaged CDH Using the Command Line.

Note: Running Services

Use the `service` command to start, stop, and restart CDH components, instead of running scripts in `/etc/init.d` directly. The `service` command creates a predictable environment by setting the current working directory to `/` and removing most environment variables (passing only *LANG* and *TERM*). With `/etc/init.d`, existing environment variables remain in force and can produce unpredictable results. When you install CDH from packages, `service` is installed as part of the Linux Standard Base (LSB).

Use the following sections to install, update, and configure HBase:

Next Steps

After installing and configuring HBase, check out the following topics about using HBase:

- Importing Data Into HBase on page 82
- Writing Data to HBase on page 80
- Reading Data from HBase on page 70

New Features and Changes for HBase in CDH 5

CDH 5.0.x and 5.1.x each include major upgrades to HBase. Each of these upgrades provides exciting new features, as well as things to keep in mind when upgrading from a previous version.

For new features and changes introduced in older CDH 5 releases, skip to CDH 5.1 HBase Changes or CDH 5.0.x HBase Changes.

CDH 5.4 HBase Changes

CDH 5.4 introduces HBase 1.0, which represents a major upgrade to HBase. This upgrade introduces new features and moves some features which were previously marked as experimental to fully supported status. This overview provides information about the most important features, how to use them, and where to find out more information. Cloudera appreciates your feedback about these features.

Highly-Available Read Replicas

CDH 5.4 introduces highly-available read replicas. Using read replicas, clients can request, on a per-read basis, a read result using a new consistency model, timeline consistency, rather than strong consistency. The read request is sent to the RegionServer serving the region, but also to any RegionServers hosting replicas of the region. The client receives

the read from the fastest RegionServer to respond, and receives an indication of whether the response was from the primary RegionServer or from a replica. See HBase Read Replicas on page 128 for more details.

MultiWAL Support

CDH 5.4 introduces support for writing multiple write-ahead logs (MultiWAL) on a given RegionServer, allowing you to increase throughput when a region writes to the WAL. See Configuring HBase MultiWAL Support on page 105.

Medium-Object (MOB) Storage

CDH 5.4 introduces a mechanism for storing objects between 100 KB and 10 MB in a default configuration, or *medium objects*, directly in HBase. Storing objects up to 50 MB is possible with additional configuration. Previously, storing these medium objects directly in HBase could degrade performance due to write amplification caused by splits and compactions.

MOB storage requires HFile V3.

doAs Impersonation for the Thrift Gateway

Prior to CDH 5.4, the Thrift gateway could be configured to authenticate to HBase on behalf of the client as a static user. A new mechanism, doAs Impersonation, allows the client to authenticate as any HBase user on a per-call basis for a higher level of security and flexibility.

Namespace Create Authorization

Prior to CDH 5.4, only global admins could create namespaces. Now, a Namespace Create authorization can be assigned to a user, who can then create namespaces.

Authorization to List Namespaces and Tables

Prior to CDH 5.4, authorization checks were not performed on list namespace and list table operations, so you could list the names of tany tables or namespaces, regardless of your authorization. In CDH 5.4, you are not able to list namespaces or tables you do not have authorization to access.

Crunch API Changes for HBase

In CDH 5.4, Apache Crunch adds the following API changes for HBase:

- `HBaseTypes.cells()` was added to support serializing HBase Cell objects.
- Each method of `HFileUtils` now supports `PCollection<C extends Cell>`, which includes both `PCollection<KeyValue>` and `PCollection<Cell>`, on their method signatures.
- `HFileTarget`, `HBaseTarget`, and `HBaseSourceTarget` each support any subclass of `Cell` as an output type. `HFileSource` and `HBaseSourceTarget` still return `KeyValue` as the input type for backward-compatibility with existing Crunch pipelines.

ZooKeeper 3.4 Is Required

HBase 1.0 requires ZooKeeper 3.4.

HBase API Changes for CDH 5.4

CDH 5.4.0 introduces HBase 1.0, which includes some major changes to the HBase APIs. Besides the changes listed above, some APIs have been deprecated in favor of new public APIs.

- The `HConnection` API is deprecated in favor of `Connection`.
- The HConnectionManager API is deprecated in favor of `ConnectionFactory`.
- The HTable API is deprecated in favor of `Table`.
- The `HTableAdmin` API is deprecated in favor of `Admin`.

HBase 1.0 API Example

```
Configuration conf = HBaseConfiguration.create();
try (Connection connection = ConnectionFactory.createConnection(conf)) {
```

```
  try (Table table = connection.getTable(TableName.valueOf(tablename)) {
    // use table as needed, the table returned is lightweight
  }
}
```

CDH 5.3 HBase Changes

CDH 5.4 introduces HBase 0.98.6, which represents a minor upgrade to HBase. CDH 5.3 provides `checkAndMutate(RowMutations)`, in addition to existing support for atomic `checkAndPut` as well as `checkAndDelete` operations on individual rows (HBASE-11796).

`SlabCache` Has Been Deprecated

`SlabCache`, which was marked as deprecated in CDH 5.2, has been removed in CDH 5.3. To configure the BlockCache, see Configuring the HBase BlockCache on page 61.

`checkAndMutate(RowMutations)` API

CDH 5.3 provides `checkAndMutate(RowMutations)`, in addition to existing support for atomic `checkAndPut` as well as `checkAndDelete` operations on individual rows (HBASE-11796).

CDH 5.2 HBase Changes

CDH 5.2 introduces HBase 0.98.6, which represents a minor upgrade to HBase. This upgrade introduces new features and moves some features which were previously marked as experimental to fully supported status. This overview provides information about the most important features, how to use them, and where to find out more information. Cloudera appreciates your feedback about these features.

`JAVA_HOME` must be set in your environment.

HBase now requires `JAVA_HOME` to be set in your environment. If it is not set, HBase will fail to start and an error will be logged. If you use Cloudera Manager, this is set automatically. If you use CDH without Cloudera Manager, `JAVA_HOME` should be set up as part of the overall installation. See Java Development Kit Installation for instructions on setting `JAVA_HOME`, as well as other JDK-specific instructions.

The default value for `hbase.hstore.flusher.count` has increased from 1 to 2.

The default value for `hbase.hstore.flusher.count` has been increased from one thread to two. This new configuration can improve performance when writing to HBase under some workloads. However, for high IO workloads two flusher threads can create additional contention when writing to HDFS. If after upgrading to CDH 5.2. you see an increase in flush times or performance degradation, lowering this value to 1 is recommended. Use the RegionServer's advanced configuration snippet for `hbase-site.xml` if you use Cloudera Manager, or edit the file directly otherwise.

The default value for `hbase.hregion.memstore.block.multiplier` has increased from 2 to 4.

The default value for `hbase.hregion.memstore.block.multiplier` has increased from 2 to 4 to improve both throughput and latency. If you experience performance degradation due to this change, change the value setting to 2, using the RegionServer's advanced configuration snippet for `hbase-site.xml` if you use Cloudera Manager, or by editing the file directly otherwise.

SlabCache is deprecated, and BucketCache is now the default block cache.

CDH 5.1 provided full support for the BucketCache block cache. CDH 5.2 deprecates usage of SlabCache in favor of BucketCache. To configure BucketCache, see BucketCache Block Cache on page 16

Changed Syntax of `user_permissions` Shell Command

The pattern-matching behavior for the `user_permissions` HBase Shell command has changed. Previously, either of the following two commands would return permissions of all known users in HBase:

```
hbase> user_permissions '*'
```

```
hbase> user_permissions '.*'
```

The first variant is no longer supported. The second variant is the only supported operation and also supports passing in other Java regular expressions.

New Properties for IPC Configuration

If the Hadoop configuration is read after the HBase configuration, Hadoop's settings can override HBase's settings if the names of the settings are the same. To avoid the risk of override, HBase has renamed the following settings (by prepending 'hbase.') so that you can set them independent of your setting for Hadoop. If you do not use the HBase-specific variants, the Hadoop settings will be used. If you have not experienced issues with your configuration, there is no need to change it.

Hadoop Configuration Property	New HBase Configuration Property
ipc.server.listen.queue.size	hbase.ipc.server.listen.queue.size
ipc.server.max.callqueue.size	hbase.ipc.server.max.callqueue.size
ipc.server.max.callqueue.length	hbase.ipc.server.max.callqueue.length
ipc.server.read.threadpool.size	hbase.ipc.server.read.threadpool.size
ipc.server.tcpkeepalive	hbase.ipc.server.tcpkeepalive
ipc.server.tcpnodelay	hbase.ipc.server.tcpnodelay
ipc.client.call.purge.timeout	hbase.ipc.client.call.purge.timeout
ipc.client.connection.maxidletime	hbase.ipc.client.connection.maxidletime
ipc.client.idlethreshold	hbase.ipc.client.idlethreshold
ipc.client.kill.max	hbase.ipc.client.kill.max

Snapshot Manifest Configuration

Snapshot manifests were previously a feature included in HBase in CDH 5 but not in Apache HBase. They are now included in Apache HBase 0.98.6. To use snapshot manifests, you now need to set hbase.snapshot.format.version to 2 in hbase-site.xml. This is the default for HBase in CDH 5.2, but the default for Apache HBase 0.98.6 is 1. To edit the configuration, use an Advanced Configuration Snippet if you use Cloudera Manager, or edit the file directly otherwise. The new snapshot code can read both version 1 and 2. However, if you use version 2, you will not be able to read these snapshots on HBase versions prior to 0.98.

Not using manifests (setting hbase.snapshot.format.version to 1) can cause excess load on the NameNode and impact performance.

Tags

Tags, which allow metadata to be stored in HFiles alongside cell data, are a feature of HFile version 3, are needed for per-cell access controls and visibility labels. Tags were previously considered an experimental feature but are now fully supported.

Per-Cell Access Controls

Per-cell access controls were introduced as an experimental feature in CDH 5.1 and are fully supported in CDH 5.2. You must use HFile version 3 to use per-cell access controls. For more information about access controls, see Per-Cell Access Controls on page 18.

Experimental Features

> **Warning:** These features are still considered experimental. Experimental features are not supported and Cloudera does not recommend using them in production environments or with important data.

Visibility Labels

You can now specify a list of visibility labels, such as CONFIDENTIAL, TOPSECRET, or PUBLIC, at the cell level. You can associate users with these labels to enforce visibility of HBase data. These labels can be grouped into complex expressions using logical operators `&`, `|`, and `!` (AND, OR, NOT). A given user is associated with a set of visibility labels, and the policy for associating the labels is pluggable. A coprocessor, `org.apache.hadoop.hbase.security.visibility.DefaultScanLabelGenerator`, checks for visibility labels on cells that would be returned by a Get or Scan and drops the cells that a user is not authorized to see, before returning the results. The same coprocessor saves visibility labels as tags, in the HFiles alongside the cell data, when a Put operation includes visibility labels. You can specify custom implementations of `ScanLabelGenerator` by setting the property `hbase.regionserver.scan.visibility.label.generator.class` to a comma-separated list of classes in `hbase-site.xml`. To edit the configuration, use an Advanced Configuration Snippet if you use Cloudera Manager, or edit the file directly otherwise.

No labels are configured by default. You can add a label to the system using either the `VisibilityClient#addLabels()` API or the `add_label` shell command. Similar APIs and shell commands are provided for deleting labels and assigning them to users. Only a user with superuser access (the `hbase.superuser` access level) can perform these operations.

To assign a visibility label to a cell, you can label the cell using the API method `Mutation#setCellVisibility(new CellVisibility(<labelExp>));`. An API is provided for managing visibility labels, and you can also perform many of the operations using HBase Shell.

Previously, visibility labels could not contain the symbols `&`, `|`, `!`, `(` and `)`, but this is no longer the case.

For more information about visibility labels, see the Visibility Labels section of the *Apache HBase Reference Guide*.

If you use visibility labels along with access controls, you must ensure that the Access Controller is loaded before the Visibility Controller in the list of coprocessors. This is the default configuration. See HBASE-11275.

Visibility labels are an **experimental** feature introduced in CDH 5.1, and still experimental in CDH 5.2.

Transparent Server-Side Encryption

Transparent server-side encryption can now be enabled for both HFiles and write-ahead logs (WALs), to protect their contents at rest. To configure transparent encryption, first create an encryption key, then configure the appropriate settings in `hbase-site.xml` . To edit the configuration, use an Advanced Configuration Snippet if you use Cloudera Manager, or edit the file directly otherwise. See the Transparent Encryption section in the *Apache HBase Reference Guide* for more information.

Transparent server-side encryption is an **experimental** feature introduced in CDH 5.1, and still experimental in CDH 5.2.

Stripe Compaction

Stripe compaction is a compaction scheme that segregates the data inside a region by row key, creating "stripes" of data which are visible within the region but transparent to normal operations. This striping improves read performance in common scenarios and greatly reduces variability by avoiding large or inefficient compactions.

Configuration guidelines and more information are available at Stripe Compaction.

To configure stripe compaction for a single table from within the HBase shell, use the following syntax.

```
alter <table>, CONFIGURATION => {<setting> => <value>}
 Example: alter 'orders', CONFIGURATION => {'hbase.store.stripe.fixed.count' => 10}
```

To configure stripe compaction for a column family from within the HBase shell, use the following syntax.

```
alter <table>, {NAME => <column family>, CONFIGURATION => {<setting => <value>}}
 Example: alter 'logs', {NAME => 'blobs', CONFIGURATION =>
{'hbase.store.stripe.fixed.count' => 10}}
```

Stripe compaction is an **experimental** feature in CDH 5.1, and still experimental in CDH 5.2.

CDH 5.1 HBase Changes

CDH 5.1 introduces HBase 0.98, which represents a major upgrade to HBase. This upgrade introduces several new features, including a section of features which are considered experimental and should not be used in a production environment. This overview provides information about the most important features, how to use them, and where to find out more information. Cloudera appreciates your feedback about these features.

In addition to HBase 0.98, Cloudera has pulled in changes from HBASE-10883, HBASE-10964, HBASE-10823, HBASE-10916, and HBASE-11275. Implications of these changes are detailed below and in the Release Notes.

BucketCache Block Cache

A new offheap BlockCache implementation, BucketCache, was introduced as an experimental feature in CDH 5 Beta 1, and is now fully supported in CDH 5.1. BucketCache can be used in either of the following two configurations:

- As a CombinedBlockCache with both onheap and offheap caches.
- As an L2 cache for the default onheap LruBlockCache

BucketCache requires less garbage-collection than SlabCache, which is the other offheap cache implementation in HBase. It also has many optional configuration settings for fine-tuning. All available settings are documented in the API documentation for CombinedBlockCache. Following is a simple example configuration.

1. First, edit `hbase-env.sh` and set `-XX:MaxDirectMemorySize` to the total size of the desired onheap plus offheap, in this case, 5 GB (but expressed as 5G). To edit the configuration, use an Advanced Configuration Snippet if you use Cloudera Manager, or edit the file directly otherwise.

```
-XX:MaxDirectMemorySize=5G
```

2. Next, add the following configuration to `hbase-site.xml`. To edit the configuration, use an Advanced Configuration Snippet if you use Cloudera Manager, or edit the file directly otherwise. This configuration uses 80% of the `-XX:MaxDirectMemorySize` (4 GB) for offheap, and the remainder (1 GB) for onheap.

```
<property>
  <name>hbase.bucketcache.ioengine</name>
  <value>offheap</value>
</property>
<property>
  <name>hbase.bucketcache.percentage.in.combinedcache</name>
  <value>0.8</value>
</property>
<property>
  <name>hbase.bucketcache.size</name>
  <value>5120</value>
</property>
```

3. Restart or rolling restart your cluster for the configuration to take effect.

Access Control for EXEC Permissions

A new access control level has been added to check whether a given user has EXEC permission. This can be specified at the level of the cluster, table, row, or cell.

To use EXEC permissions, perform the following procedure.

- Install the AccessController coprocessor either as a system coprocessor or on a table as a table coprocessor.

- Set the `hbase.security.exec.permission.checks` configuration setting in `hbase-site.xml` to `true` To edit the configuration, use an Advanced Configuration Snippet if you use Cloudera Manager, or edit the file directly otherwise..

For more information on setting and revoking security permissions, see the Access Control section of the *Apache HBase Reference Guide*.

Reverse Scan API

A reverse scan API has been introduced. This allows you to scan a table in reverse. Previously, if you wanted to be able to access your data in either direction, you needed to store the data in two separate tables, each ordered differently. This feature was implemented in HBASE-4811.

To use the reverse scan feature, use the new `Scan.setReversed(boolean reversed)` API. If you specify a startRow and stopRow, to scan in reverse, the startRow needs to be lexicographically after the stopRow. See the Scan API documentation for more information.

MapReduce Over Snapshots

You can now run a MapReduce job over a snapshot from HBase, rather than being limited to live data. This provides the ability to separate your client-side work load from your live cluster if you need to run resource-intensive MapReduce jobs and can tolerate using potentially-stale data. You can either run the MapReduce job on the snapshot within HBase, or export the snapshot and run the MapReduce job against the exported file.

Running a MapReduce job on an exported file outside of the scope of HBase relies on the permissions of the underlying filesystem and server, and bypasses ACLs, visibility labels, and encryption that may otherwise be provided by your HBase cluster.

A new API, `TableSnapshotInputFormat`, is provided. For more information, see TableSnapshotInputFormat.

MapReduce over snapshots was introduced in CDH 5.0.

Stateless Streaming Scanner over REST

A new stateless streaming scanner is available over the REST API. Using this scanner, clients do not need to restart a scan if the REST server experiences a transient failure. All query parameters are specified during the REST request. Query parameters include `startrow`, `endrow`, `columns`, `starttime`, `endtime`, `maxversions`, `batchtime`, and limit. Following are a few examples of using the stateless streaming scanner.

Scan the entire table, return the results in JSON.

```
curl -H "Accept: application/json" https://localhost:8080/ExampleScanner/*
```

Scan the entire table, return the results in XML.

```
curl -H "Content-Type: text/xml" https://localhost:8080/ExampleScanner/*
```

Scan only the first row.

```
curl -H "Content-Type: text/xml" \
https://localhost:8080/ExampleScanner/*?limit=1
```

Scan only specific columns.

```
curl -H "Content-Type: text/xml" \
https://localhost:8080/ExampleScanner/*?columns=a:1,b:1
```

Scan for rows between starttime and endtime.

```
curl -H "Content-Type: text/xml" \
https://localhost:8080/ExampleScanner/*?starttime=1389900769772\
&endtime=1389900800000
```

Scan for a given row prefix.

```
curl -H "Content-Type: text/xml" https://localhost:8080/ExampleScanner/test*
```

For full details about the stateless streaming scanner, see the API documentation for this feature.

Delete Methods of Put Class Now Use Constructor Timestamps

The `Delete()` methods of the `Put` class of the HBase Client API previously ignored the constructor's timestamp, and used the value of `HConstants.LATEST_TIMESTAMP`. This behavior was different from the behavior of the `add()` methods. The `Delete()` methods now use the timestamp from the constructor, creating consistency in behavior across the `Put` class. See HBASE-10964.

Experimental Features

> **Warning:** These features are still considered experimental. Experimental features are not supported and Cloudera does not recommend using them in production environments or with important data.

Visibility Labels

You can now specify a list of visibility labels, such as CONFIDENTIAL, TOPSECRET, or PUBLIC, at the cell level. You can associate users with these labels to enforce visibility of HBase data. These labels can be grouped into complex expressions using logical operators `&`, `|`, and `!` (AND, OR, NOT). A given user is associated with a set of visibility labels, and the policy for associating the labels is pluggable. A coprocessor, `org.apache.hadoop.hbase.security.visibility.DefaultScanLabelGenerator`, checks for visibility labels on cells that would be returned by a Get or Scan and drops the cells that a user is not authorized to see, before returning the results. The same coprocessor saves visibility labels as tags, in the HFiles alongside the cell data, when a Put operation includes visibility labels. You can specify custom implementations of `ScanLabelGenerator` by setting the property `hbase.regionserver.scan.visibility.label.generator.class` to a comma-separated list of classes.

No labels are configured by default. You can add a label to the system using either the `VisibilityClient#addLabels()` API or the `add_label` shell command. Similar APIs and shell commands are provided for deleting labels and assigning them to users. Only a user with superuser access (the `hbase.superuser` access level) can perform these operations.

To assign a visibility label to a cell, you can label the cell using the API method `Mutation#setCellVisibility(new CellVisibility(<labelExp>));`.

Visibility labels and request authorizations cannot contain the symbols `&`, `|`, `!`, `(` and `)` because they are reserved for constructing visibility expressions. See HBASE-10883.

For more information about visibility labels, see the Visibility Labels section of the *Apache HBase Reference Guide*.

If you use visibility labels along with access controls, you must ensure that the Access Controller is loaded before the Visibility Controller in the list of coprocessors. This is the default configuration. See HBASE-11275.

To use per-cell access controls or visibility labels, you must use HFile version 3. To enable HFile version 3, add the following to `hbase-site.xml`, using an advanced code snippet if you use Cloudera Manager, or directly to the file if your deployment is unmanaged.. Changes will take effect after the next major compaction.

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

Visibility labels are an **experimental** feature introduced in CDH 5.1.

Per-Cell Access Controls

You can now specify access control levels at the per-cell level, as well as at the level of the cluster, table, or row.

A new parent class has been provided, which encompasses `Get`, `Scan`, and `Query`. This change also moves the `getFilter` and `setFilter` methods of `Get` and `Scan` to the common parent class. Client code may need to be recompiled to take advantage of per-cell ACLs. See the Access Control section of the *Apache HBase Reference Guide* for more information.

The ACLS for cells having timestamps in the future are not considered for authorizing the pending mutation operations. See HBASE-10823.

If you use visibility labels along with access controls, you must ensure that the Access Controller is loaded before the Visibility Controller in the list of coprocessors. This is the default configuration.

To use per-cell access controls or visibility labels, you must use HFile version 3. To enable HFile version 3, add the following to `hbase-site.xml`, using an advanced code snippet if you use Cloudera Manager, or directly to the file if your deployment is unmanaged.. Changes will take effect after the next major compaction.

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

Per-cell access controls are an **experimental** feature introduced in CDH 5.1.

Transparent Server-Side Encryption

Transparent server-side encryption can now be enabled for both HFiles and write-ahead logs (WALs), to protect their contents at rest. To configure transparent encryption, first create an encryption key, then configure the appropriate settings in `hbase-site.xml` . See the Transparent Encryption section in the *Apache HBase Reference Guide* for more information.

Transparent server-side encryption is an **experimental** feature introduced in CDH 5.1.

Stripe Compaction

Stripe compaction is a compaction scheme that segregates the data inside a region by row key, creating "stripes" of data which are visible within the region but transparent to normal operations. This striping improves read performance in common scenarios and greatly reduces variability by avoiding large or inefficient compactions.

Configuration guidelines and more information are available at Stripe Compaction.

To configure stripe compaction for a single table from within the HBase shell, use the following syntax.

```
alter <table>, CONFIGURATION => {<setting> => <value>}
 Example: alter 'orders', CONFIGURATION => {'hbase.store.stripe.fixed.count' => 10}
```

To configure stripe compaction for a column family from within the HBase shell, use the following syntax.

```
alter <table>, {NAME => <column family>, CONFIGURATION => {<setting => <value>}}
 Example: alter 'logs', {NAME => 'blobs', CONFIGURATION =>
{'hbase.store.stripe.fixed.count' => 10}}
```

Stripe compaction is an **experimental** feature in CDH 5.1.

Distributed Log Replay

After a RegionServer fails, its failed region is assigned to another RegionServer, which is marked as "recovering" in ZooKeeper. A SplitLogWorker directly replays edits from the WAL of the failed RegionServer to the region at its new location. When a region is in "recovering" state, it can accept writes but no reads (including Append and Increment), region splits or merges. Distributed Log Replay extends the distributed log splitting framework. It works by directly replaying WAL edits to another RegionServer instead of creating `recovered.edits` files.

Distributed log replay provides the following advantages over using the current distributed log splitting functionality on its own.

- It eliminates the overhead of writing and reading a large number of `recovered.edits` files. It is not unusual for thousands of recovered.edits files to be created and written concurrently during a RegionServer recovery. Many small random writes can degrade overall system performance.

- It allows writes even when a region is in recovering state. It only takes seconds for a recovering region to accept writes again.

To enable distributed log replay, set `hbase.master.distributed.log.replay` to `true`. You must also enable HFile version 3. Distributed log replay is unsafe for rolling upgrades.

Distributed log replay is an **experimental** feature in CDH 5.1.

CDH 5.0.x HBase Changes

HBase in CDH 5.0.x is based on the Apache HBase 0.96 release. When upgrading to CDH 5.0.x, keep the following in mind.

Upgrade is Not Reversible

The upgrade from CDH 4 HBase to CDH 5 HBase is irreversible and requires HBase to be shut down completely. Executing the upgrade script reorganizes existing HBase data stored on HDFS into new directory structures, converts HBase 0.90 HFile v1 files to the improved and optimized HBase 0.96 HFile v2 file format, and rewrites the `hbase.version` file. This upgrade also removes transient data stored in ZooKeeper during the conversion to the new data format.

These changes were made to reduce the impact in future major upgrades. Previously HBase used brittle custom data formats and this move shifts HBase's RPC and persistent data to a more evolvable Protocol Buffer data format.

API Changes

The HBase User API (Get, Put, Result, Scanner etc; see Apache HBase API documentation) has evolved and attempts have been made to make sure the HBase Clients are source code compatible and thus should recompile without needing any source code modifications. This cannot be guaranteed however, since with the conversion to Protocol Buffers (ProtoBufs), some relatively obscure APIs have been removed. Rudimentary efforts have also been made to preserve recompile compatibility with advanced APIs such as Filters and Coprocessors. These advanced APIs are still evolving and our guarantees for API compatibility are weaker here.

For information about changes to custom filters, see Custom Filters.

As of 0.96, the User API has been marked and all attempts at compatibility in future versions will be made. A version of the javadoc that only contains the User API can be found here.

HBase Metrics Changes

HBase provides a metrics framework based on JMX beans. Between HBase 0.94 and 0.96, the metrics framework underwent many changes. Some beans were added and removed, some metrics were moved from one bean to another, and some metrics were renamed or removed. Click here to download the CSV spreadsheet which provides a mapping.

Custom Filters

If you used custom filters written for HBase 0.94, you need to recompile those filters for HBase 0.96. The custom filter must be altered to fit with the newer interface that uses protocol buffers. Specifically two new methods, `toByteArray(…)` and `parseFrom(…)`, which are detailed in detailed in the Filter API. These should be used instead of the old methods `write(…)` and `readFields(…)`, so that protocol buffer serialization is used. To see what changes were required to port one of HBase's own custom filters, see the Git commit that represented porting the `SingleColumnValueFilter` filter.

Checksums

When you upgrade to CDH 5, HBase checksums are now turned on by default. With this configuration, HBase reads data and then verifies the checksums. Checksum verification inside HDFS will be switched off. If the HBase-checksum verification fails, then the HDFS checksums are used instead for verifying data that is being read from storage. Once you turn on HBase checksums, you will not be able to roll back to an earlier HBase version.

You should see a modest performance gain after setting `hbase.regionserver.checksum.verify` to true for data that is not already present in the RegionServer's block cache.

To enable or disable checksums, modify the following configuration properties in `hbase-site.xml`. To edit the configuration, use an Advanced Configuration Snippet if you use Cloudera Manager, or edit the file directly otherwise.

```
<property>
  <name>hbase.regionserver.checksum.verify</name>
  <value>true</value>
  <description>
      If set to  true, HBase will read data and then verify checksums  for
      hfile blocks. Checksum verification inside HDFS will be switched off.
      If the hbase-checksum verification fails, then it will  switch back to
      using HDFS checksums.
  </description>
</property>
```

The default value for the `hbase.hstore.checksum.algorithm` property has also changed to `CRC32`. Previously, Cloudera advised setting it to `NULL` due to performance issues which are no longer a problem.

```
<property>
   <name>hbase.hstore.checksum.algorithm</name>
   <value>CRC32</value>
   <description>
     Name of an algorithm that is used to compute checksums. Possible values
     are NULL, CRC32, CRC32C.
   </description>
 </property>
```

Installing HBase

To install HBase On RHEL-compatible systems:

```
$ sudo yum install hbase
```

To install HBase on Ubuntu and Debian systems:

```
$ sudo apt-get install hbase
```

To install HBase on SLES systems:

```
$ sudo zypper install hbase
```

Note: See also Starting HBase in Standalone Mode on page 22, Configuring HBase in Pseudo-Distributed Mode on page 24, and Deploying HBase on a Cluster on page 26 for more information on configuring HBase for different modes.

To list the installed files on Ubuntu and Debian systems:

```
$ dpkg -L hbase
```

To list the installed files on RHEL and SLES systems:

```
$ rpm -ql hbase
```

You can see that the HBase package has been configured to conform to the Linux Filesystem Hierarchy Standard. (To learn more, run `man hier`).

You are now ready to enable the server daemons you want to use with Hadoop. You can also enable Java-based client access by adding the JAR files in `/usr/lib/hbase/` and `/usr/lib/hbase/lib/` to your Java class path.

Starting HBase in Standalone Mode

> **Note:**
>
> You can skip this section if you are already running HBase in distributed or pseudo-distributed mode.

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.

> **Important:**
>
> If you have configured High Availability for the NameNode (HA), you cannot deploy HBase in standalone mode without modifying the default configuration, because both the standalone HBase process and ZooKeeper (required by HA) will try to bind to port 2181. You can configure a different port for ZooKeeper, but in most cases it makes more sense to deploy HBase in distributed mode in an HA cluster.

In order to run HBase in standalone mode, you must install the HBase Master package.

Installing the HBase Master

To install the HBase Master on RHEL-compatible systems:

```
$ sudo yum install hbase-master
```

To install the HBase Master on Ubuntu and Debian systems:

```
$ sudo apt-get install hbase-master
```

To install the HBase Master on SLES systems:

```
$ sudo zypper install hbase-master
```

Starting the HBase Master

- On RHEL and SLES systems (using `.rpm` packages) you can now start the HBase Master by using the included service script:

```
$ sudo service hbase-master start
```

- On Ubuntu systems (using Debian packages) the HBase Master starts when the HBase package is installed.

To verify that the standalone installation is operational, visit `http://localhost:60010`. The list of RegionServers at the bottom of the page should include one entry for your local machine.

> **Note:**
>
> Although you have only started the master process, in *standalone* mode this same process is also internally running a RegionServer and a ZooKeeper peer. In the next section, you will break out these components into separate JVMs.

If you see this message when you start the HBase standalone master:

```
Starting Hadoop HBase master daemon: starting master, logging to
/usr/lib/hbase/logs/hbase-hbase-master/cloudera-vm.out
Couldnt start ZK at requested address of 2181, instead got: 2182.  Aborting. Why? Because
 clients (eg shell) wont be able to find this ZK quorum
hbase-master.
```

you will need to stop the hadoop-zookeeper-server (or zookeeper-server) or uninstall the hadoop-zookeeper-server (or zookeeper) package.

See also Accessing HBase by using the HBase Shell on page 27, Using MapReduce with HBase on page 29 and Troubleshooting HBase on page 29.

Installing and Starting the HBase Thrift Server

To install Thrift on RHEL-compatible systems:

```
$ sudo yum install hbase-thrift
```

To install Thrift on Ubuntu and Debian systems:

```
$ sudo apt-get install hbase-thrift
```

To install Thrift on SLES systems:

```
$ sudo zypper install hbase-thrift
```

You can now use the `service` command to start the Thrift server:

```
$ sudo service hbase-thrift start
```

Installing and Configuring HBase REST

To install HBase REST on RHEL-compatible systems:

```
$ sudo yum install hbase-rest
```

To install HBase REST on Ubuntu and Debian systems:

```
$ sudo apt-get install hbase-rest
```

To install HBase REST on SLES systems:

```
$ sudo zypper install hbase-rest
```

You can use the `service` command to run an `init.d` script, `/etc/init.d/hbase-rest`, to start the REST server; for example:

```
$ sudo service hbase-rest start
```

The script starts the server by default on port 8080. This is a commonly used port and so may conflict with other applications running on the same host.

If you need change the port for the REST server, configure it in `hbase-site.xml`, for example:

```
<property>
  <name>hbase.rest.port</name>
  <value>60050</value>
</property>
```

Note:

You can use HBASE_REST_OPTS in `hbase-env.sh` to pass other settings (such as heap size and GC parameters) to the REST server JVM.

Configuring HBase in Pseudo-Distributed Mode

Note: You can skip this section if you are already running HBase in distributed mode, or if you intend to use only standalone mode.

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. This section also assumes you want to store your HBase data in HDFS rather than on the local filesystem.

Note: Before you start

- This section assumes you have already installed the HBase master and gone through the standalone configuration steps.
- If the HBase master is already running in standalone mode, use the following command to stop it:

```
sudo service hbase-master stop
```

Modifying the HBase Configuration

To enable pseudo-distributed mode, you must first make some configuration changes. Open `/etc/hbase/conf/hbase-site.xml` in your editor of choice, and insert the following XML properties between the `<configuration>` and `</configuration>` tags. The `hbase.cluster.distributed` property directs HBase to start each process in a separate JVM. The `hbase.rootdir` property directs HBase to store its data in an HDFS filesystem, rather than the local filesystem. Be sure to replace `myhost` with the hostname of your HDFS NameNode (as specified by `fs.default.name` or `fs.defaultFS` in your `conf/core-site.xml` file); you may also need to change the port number from the default (8020).

```
<property>
  <name>hbase.cluster.distributed</name>
  <value>true</value>
</property>
<property>
  <name>hbase.rootdir</name>
  <value>hdfs://myhost:8020/hbase</value>
</property>
```

Creating the /hbase Directory in HDFS

Before starting the HBase Master, you need to create the `/hbase` directory in HDFS. The HBase master runs as `hbase:hbase` so it does not have the required permissions to create a top level directory.

To create the /hbase directory in HDFS:

```
$ sudo -u hdfs hadoop fs -mkdir /hbase
$ sudo -u hdfs hadoop fs -chown hbase /hbase
```

Note: If Kerberos is enabled, do not use commands in the form `sudo -u <user> hadoop <command>`; they will fail with a security error. Instead, use the following commands: `$ kinit <user>` (if you are using a password) *or* `$ kinit -kt <keytab> <principal>` (if you are using a `keytab`) and then, for each command executed by this user, `$ <command>`

Enabling Servers for Pseudo-distributed Operation

After you have configured HBase, you must enable the various servers that make up a distributed HBase cluster. HBase uses three required types of servers:

- Installing and Starting ZooKeeper Server
- Starting the HBase Master
- Starting an HBase RegionServer

Installing and Starting ZooKeeper Server

HBase uses ZooKeeper Server as a highly available, central location for cluster management. For example, it allows clients to locate the servers, and ensures that only one master is active at a time. For a small cluster, running a ZooKeeper node collocated with the NameNode is recommended. For larger clusters, contact Cloudera Support for configuration help.

Install and start the ZooKeeper Server in standalone mode by running the commands shown in the Installing the ZooKeeper Server Package and Starting ZooKeeper on a Single Server

Starting the HBase Master

After ZooKeeper is running, you can start the HBase master in standalone mode.

```
$ sudo service hbase-master start
```

Starting an HBase RegionServer

The RegionServer is the HBase process that actually hosts data and processes requests. The RegionServer typically runs on all HBase nodes except for the node running the HBase master node.

To enable the HBase RegionServer On RHEL-compatible systems:

```
$ sudo yum install hbase-regionserver
```

To enable the HBase RegionServer on Ubuntu and Debian systems:

```
$ sudo apt-get install hbase-regionserver
```

To enable the HBase RegionServer on SLES systems:

```
$ sudo zypper install hbase-regionserver
```

To start the RegionServer:

```
$ sudo service hbase-regionserver start
```

Verifying the Pseudo-Distributed Operation

After you have started ZooKeeper, the Master, and a RegionServer, the pseudo-distributed cluster should be up and running. You can verify that each of the daemons is running using the `jps` tool from the Oracle JDK, which you can

obtain from here. If you are running a pseudo-distributed HDFS installation and a pseudo-distributed HBase installation on one machine, `jps` will show the following output:

```
$ sudo jps
32694 Jps
30674 HRegionServer
29496 HMaster
28781 DataNode
28422 NameNode
30348 QuorumPeerMain
```

You should also be able to go to `http://localhost:60010` and verify that the local RegionServer has registered with the Master.

Installing and Starting the HBase Thrift Server

The HBase Thrift Server is an alternative gateway for accessing the HBase server. Thrift mirrors most of the HBase client APIs while enabling popular programming languages to interact with HBase. The Thrift Server is multiplatform and more performant than REST in many situations. Thrift can be run collocated along with the RegionServers, but should not be collocated with the NameNode or the JobTracker. For more information about Thrift, visit http://thrift.apache.org/.

To enable the HBase Thrift Server On RHEL-compatible systems:

```
$ sudo yum install hbase-thrift
```

To enable the HBase Thrift Server on Ubuntu and Debian systems:

```
$ sudo apt-get install hbase-thrift
```

To enable the HBase Thrift Server on SLES systems:

```
$ sudo zypper install hbase-thrift
```

To start the Thrift server:

```
$ sudo service hbase-thrift start
```

See also Accessing HBase by using the HBase Shell on page 27, Using MapReduce with HBase on page 29 and Troubleshooting HBase on page 29.

Deploying HBase on a Cluster

After you have HBase running in pseudo-distributed mode, the same configuration can be extended to running on a distributed cluster.

Note: Before you start

This section assumes that you have already installed the HBase Master and the HBase RegionServer and gone through the steps for standalone and pseudo-distributed configuration. You are now about to distribute the processes across multiple hosts; see Choosing Where to Deploy the Processes on page 26.

Choosing Where to Deploy the Processes

For small clusters, Cloudera recommends designating one node in your cluster as the HBase Master node. On this node, you will typically run the HBase Master and a ZooKeeper quorum peer. These master processes may be collocated with the Hadoop NameNode and JobTracker for small clusters.

Designate the remaining nodes as RegionServer nodes. On each node, Cloudera recommends running a RegionServer, which may be collocated with a Hadoop TaskTracker (MRv1) and a DataNode. When co-locating with TaskTrackers, be sure that the resources of the machine are not oversubscribed – it's safest to start with a small number of MapReduce slots and work up slowly.

The HBase Thrift service is light-weight, and can be run on any node in the cluster.

Configuring for Distributed Operation

After you have decided which machines will run each process, you can edit the configuration so that the nodes can locate each other. In order to do so, you should make sure that the configuration files are synchronized across the cluster. Cloudera strongly recommends the use of a configuration management system to synchronize the configuration files, though you can use a simpler solution such as `rsync` to get started quickly.

The only configuration change necessary to move from pseudo-distributed operation to fully-distributed operation is the addition of the ZooKeeper Quorum address in `hbase-site.xml`. Insert the following XML property to configure the nodes with the address of the node where the ZooKeeper quorum peer is running:

```
<property>
  <name>hbase.zookeeper.quorum</name>
  <value>mymasternode</value>
</property>
```

The `hbase.zookeeper.quorum` property is a comma-separated list of hosts on which ZooKeeper servers are running. If one of the ZooKeeper servers is down, HBase will use another from the list. By default, the ZooKeeper service is bound to port 2181. To change the port, add the `hbase.zookeeper.property.clientPort` property to `hbase-site.xml` and set the value to the port you want ZooKeeper to use. In CDH 5.7.0 and higher, you do not need to use hbase.zookeeper.property.clientPort. Instead, you can specify the port along with the hostname for each ZooKeeper host:

```
<property>
  <name>hbase.zookeeper.quorum</name>
  <value>zk1.example.com:2181,zk2.example.com:20000,zk3.example.com:31111</value>
</property>
```

For more information, see this chapter of the Apache HBase Reference Guide.

To start the cluster, start the services in the following order:

1. The ZooKeeper Quorum Peer
2. The HBase Master
3. Each of the HBase RegionServers

After the cluster is fully started, you can view the HBase Master web interface on port 60010 and verify that each of the RegionServer nodes has registered properly with the master.

See also Accessing HBase by using the HBase Shell on page 27, Using MapReduce with HBase on page 29 and Troubleshooting HBase on page 29. For instructions on improving the performance of local reads, see Optimizing Performance in CDH.

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.

Configuring HBase Online Merge

CDH 5 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.

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 1: 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

Location	Purpose	Troubleshooting Notes
		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.

Upgrading HBase

Coprocessors and Custom JARs

When upgrading HBase from one major version to another, you must recompile coprocessors and custom JARs *after* the upgrade.

Never rely on HBase directory layout on disk.

The HBase directory layout is an implementation detail and is subject to change. Do not rely on the directory layout for client or administration functionality. Instead, access HBase using the supported APIs.

Upgrading HBase from a Lower CDH 5 Release

Important: Rolling upgrade is not supported between a CDH 5 Beta release and a CDH 5 GA release. Cloudera recommends using Cloudera Manager if you need to do rolling upgrades.

To upgrade HBase from a lower CDH 5 release, proceed as follows.

The instructions that follow assume that you are upgrading HBase as part of an upgrade to the latest CDH 5 release, and have already performed the steps underUpgrading from an Earlier CDH 5 Release to the Latest Release.

During a rolling upgrade from CDH 5.0.x to CDH 5.4.x the HBase Master UI will display the URLs to the old HBase RegionServers using an incorrect info port number. Once the rolling upgrade completes the HBase master UI will use the correct port number.

Step 1: Perform a Graceful Cluster Shutdown

Note: Upgrading using rolling restart is not supported.

To shut HBase down gracefully:

1. Stop the Thrift server and clients, then stop the cluster.
   a. Stop the Thrift server and clients:

```
sudo service hbase-thrift stop
```

   b. Stop the cluster by shutting down the master and the RegionServers:
      - Use the following command on the master node:

```
sudo service hbase-master stop
```

      - Use the following command on each node hosting a RegionServer:

```
sudo service hbase-regionserver stop
```

2. Stop the ZooKeeper Server:

```
$ sudo service zookeeper-server stop
```

Step 2: Install the new version of HBase

Note: You may want to take this opportunity to upgrade ZooKeeper, but you do not *have* to upgrade Zookeeper before upgrading HBase; the new version of HBase will run with the older version of Zookeeper. For instructions on upgrading ZooKeeper, see Upgrading ZooKeeper from an Earlier CDH 5 Release.

To install the new version of HBase, follow directions in the next section, HBase Installation on page 11.

Important: Configuration files

- If you install a newer version of a package that is already on the system, configuration files that you have modified will remain intact.
- If you uninstall a package, the package manager renames any configuration files you have modified from `<file>` to `<file>.rpmsave`. If you then re-install the package (probably to install a new version) the package manager creates a new `<file>` with applicable defaults. You are responsible for applying any changes captured in the original configuration file to the new configuration file. In the case of Ubuntu and Debian upgrades, you will be prompted if you have made changes to a file for which there is a new version. For details, see Automatic handling of configuration files by `dpkg`.

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. 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 5.14.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 5.14.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 61 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.

Configuring Cell Level TTL in HBase

Cell TTLs are defined internally as Cell Tags. Cell Tags are only supported for HFile Version 3 and higher, therefore HFile Version 3 must be set to enable Cell TTL use. For more information, see Enabling HFile Version 3 Using Clouder Manager.

Using Hedged Reads

Important:

- Follow these command-line instructions on systems that do not use Cloudera Manager.
- This information applies specifically to CDH 5.14.x. See Cloudera Documentation for information specific to other releases.

Note:

To enable hedged reads for HBase, edit the `hbase-site.xml` file on each server. Set `dfs.client.hedged.read.threadpool.size` to the number of threads to dedicate to running hedged threads, and set the `dfs.client.hedged.read.threshold.millis` configuration property to the number of milliseconds to wait before starting a second read against a different block replica. Set `dfs.client.hedged.read.threadpool.size` to 0 or remove it from the configuration to disable the feature. After changing these properties, restart your cluster.

The following is an example configuration for hedged reads for HBase.

```
<property>
  <name>dfs.client.hedged.read.threadpool.size</name>
  <value>20</value>  <!-- 20 threads -->
</property>
<property>
  <name>dfs.client.hedged.read.threshold.millis</name>
  <value>10</value>  <!-- 10 milliseconds -->
</property>
```

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.

Configuring HBase Online Merge

CDH 5 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. 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. Type `add_rsgroup '*mygroup*'`.

2. Add RegionServers and tables to this rsgroup using the commands:

```
hbase> move_servers_rsgroup 'mygroup',['server1:port','server2:port']
hbase> move_tables_rsgroup 'mygroup',['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
```

Configuring the BlockCache

See Configuring the HBase BlockCache on page 61.

Configuring the Scanner Heartbeat

See Configuring the HBase Scanner Heartbeat on page 68.

Troubleshooting HBase

See Troubleshooting HBase.

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**)

> **Note:** End-user custom HBase coprocessors are not supported.

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

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.

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**)

> **Note:** End-user custom HBase coprocessors are not supported.

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

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 46.
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**.

Starting and Stopping HBase Using the Command Line

When starting and stopping CDH services, order is important. See Starting CDH Services Using the Command Line and Stopping CDH Services Using the Command Line for details. If you use Cloudera Manager, follow these instructions instead.

Important:

- If you use Cloudera Manager, do not use these command-line instructions.
- This information applies specifically to CDH 5.14.x. If you use an earlier version of CDH, see the documentation for that version located at Cloudera Documentation.

Starting HBase

When starting HBase, it is important to start the HMaster, followed by the RegionServers, then the Thrift server.

If you need the ability to perform a rolling restart, Cloudera recommends managing your cluster with Cloudera Manager.

1. To start a HBase cluster using the command line, start the HBase Master by using the `sudo hbase-master start` command on RHEL or SuSE, or the `sudo hadoop-hbase-regionserver start` command on Ubuntu or Debian. The HMaster starts the RegionServers automatically.
2. To start a RegionServer manually, use the `sudo hbase-regionserver start` command on RHEL or SuSE, or the `sudo hadoop-hbase-regionserver start` command on Ubuntu or Debian.
3. The Thrift service has no dependencies and can be restarted at any time. To start the Thrift server, use the `hbase-thrift start` on RHEL or SuSE, or the `hadoop-hbase-thrift start` on Ubuntu or Debian.

Stopping HBase

When stopping HBase, it is important to stop the Thrift server, followed by each RegionServer, followed by any backup HMasters, and finally the main HMaster.

1. Shut down the Thrift server by using the `hbase-thrift stop` command on the Thrift server host. `sudo service hbase-thrift stop`
2. Shut down each RegionServer by using the `hadoop-hbase-regionserver stop` command on the RegionServer host.

```
sudo service hadoop-hbase-regionserver stop
```

3. Shut down backup HMasters, followed by the main HMaster, by using the `hbase-master stop` command.

```
sudo service hbase-master 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 speicfied, 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.

```
$ bin/hbase hbck -h

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.
   -disableBalancer Disable the load balancer.

  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.
   -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
   -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.
   -fixVersionFile   Try to fix missing hbase.version file 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.
   -ignorePreCheckPermission  ignore filesystem permission pre-check
   -fixReferenceFiles  Try to offline lingering reference store files
   -fixEmptyMetaCells  Try to fix hbase:meta entries not referencing any region (empty
REGIONINFO_QUALIFIER rows)

  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

  Metadata Repair shortcuts
   -repair           Shortcut for -fixAssignments -fixMeta -fixHdfsHoles
                                  -fixHdfsOrphans -fixHdfsOverlaps -fixVersionFile
                                  -sidelineBigOverlaps -fixReferenceFiles -fixTableLocks
                                  -fixOrphanedTableZnodes
   -repairHoles      Shortcut for -fixAssignments -fixMeta -fixHdfsHoles

  Table lock options
   -fixTableLocks    Deletes table locks held for a long time (hbase.table.lock.expire.ms,
                     10min by default)

  Table Znode options
   -fixOrphanedTableZnodes    Set table state in ZNode to disabled if table does not exists

 Replication options
   -fixReplication   Deletes replication queues for removed peers
```

`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.
```

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.

If your cluster is managed by Cloudera Manager, follow the instructions in Configure HBase Garbage Collection Using Cloudera Manager on page 56. Otherwise, use Configure HBase Garbage Collection Using the Command Line on page 56.

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. Click **Save Changes** to commit the changes.
8. Restart the role.

Configure HBase Garbage Collection Using the Command Line

Important:

- Follow these command-line instructions on systems that do not use Cloudera Manager.
- This information applies specifically to CDH 5.14.x. See Cloudera Documentation for information specific to other releases.

1. On each RegionServer, edit `conf/hbase-env.sh`.
2. Add or modify JVM configuration options on the line beginning with `HBASE_OPTS`.
3. Restart the RegionServer.

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 in CDH 5.7 and higher. 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. Click **Save Changes** to commit the changes.
8. Restart the service.

Disable the `BoundedByteBufferPool` Using the Command Line

Important:

- Follow these command-line instructions on systems that do not use Cloudera Manager.
- This information applies specifically to CDH 5.14.x. See Cloudera Documentation for information specific to other releases.

1. On each RegionServer, edit `conf/hbase-site.xml`.
2. Add the following XML:

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

3. Save your changes.
4. Restart the RegionServer.

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. Click **Save Changes** to commit the changes.
9. Restart the role.
10. Restart the service.

Configure the HBase Canary Using the Command Line

Important:

- Follow these command-line instructions on systems that do not use Cloudera Manager.
- This information applies specifically to CDH 5.14.x. See Cloudera Documentation for information specific to other releases.

The HBase canary is a Java class. To run it from the command line, in the foreground, issue a command similar to the following, as the HBase user:

```
$ /usr/bin/hbase org.apache.hadoop.hbase.tool.Canary
```

To start the canary in the background, add the `--daemon` option. You can also use this option in your HBase startup scripts.

```
$ /usr/bin/hbase org.apache.hadoop.hbase.tool.Canary --daemon
```

The canary has many options. To see usage instructions, add the `--help` parameter:

```
$ /usr/bin/hbase org.apache.hadoop.hbase.tool.Canary --help
```

Checking and Repairing HBase Tables

HBaseFsck (`hbck`) is a command-line tool that checks for region consistency and table integrity problems and repairs corruption. It works in two basic modes — a read-only inconsistency identifying mode and a multi-phase read-write repair mode.

- **Read-only inconsistency identification**: In this mode, which is the default, a report is generated but no repairs are attempted.
- **Read-write repair mode**: In this mode, if errors are found, hbck attempts to repair them.

Always run HBase administrative commands such as the HBase Shell, `hbck`, or bulk-load commands as the HBase user (typically `hbase`).

Running `hbck` Manually

The `hbck` command is located in the bin directory of the HBase install.

- With no arguments, `hbck` checks HBase for inconsistencies and prints OK if no inconsistencies are found, or the number of inconsistencies otherwise.
- With the `-details` argument, `hbck` checks HBase for inconsistencies and prints a detailed report.
- To limit `hbck` to only checking specific tables, provide them as a space-separated list: `hbck <table1> <table2>`

Warning: The following `hbck` options modify HBase metadata and are dangerous. They are not coordinated by the HMaster and can cause further corruption by conflicting with commands that are currently in progress or coordinated by the HMaster. Even if the HMaster is down, it may try to recover the latest operation when it restarts. These options should only be used as a last resort. The `hbck` command can only fix actual HBase metadata corruption and is not a general-purpose maintenance tool. Before running these commands, consider contacting Cloudera Support for guidance. In addition, running any of these commands requires an HMaster restart.

- If region-level inconsistencies are found, use the `-fix` argument to direct `hbck` to try to fix them. The following sequence of steps is followed:
  1. The standard check for inconsistencies is run.
  2. If needed, repairs are made to tables.
  3. If needed, repairs are made to regions. Regions are closed during repair.
- You can also fix individual region-level inconsistencies separately, rather than fixing them automatically with the `-fix` argument.
  - `-fixAssignments` repairs unassigned, incorrectly assigned or multiply assigned regions.
  - `-fixMeta` removes rows from `hbase:meta` when their corresponding regions are not present in HDFS and adds new meta rows if regions are present in HDFS but not in `hbase:meta`.
  - `-repairHoles` creates HFiles for new empty regions on the filesystem and ensures that the new regions are consistent.
  - `-fixHdfsOrphans` repairs a region directory that is missing a region metadata file (the `.regioninfo` file).
  - `-fixHdfsOverlaps` fixes overlapping regions. You can further tune this argument using the following options:
    - `-maxMerge` <*n*> controls the maximum number of regions to merge.
    - `-sidelineBigOverlaps` attempts to sideline the regions which overlap the largest number of other regions.
    - `-maxOverlapsToSideline` <*n*> limits the maximum number of regions to sideline.
- To try to repair all inconsistencies and corruption at once, use the `-repair` option, which includes all the region and table consistency options.

For more details about the `hbck` command, see Appendix C of the HBase Reference Guide.

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

Enabling Hedged Reads 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 5.14.x. See Cloudera Documentation for information specific to other releases.

To enable hedged reads for HBase, edit the `hbase-site.xml` file on each server. Set `dfs.client.hedged.read.threadpool.size` to the number of threads to dedicate to running hedged threads, and set the `dfs.client.hedged.read.threshold.millis` configuration property to the number of milliseconds to wait before starting a second read against a different block replica. Set `dfs.client.hedged.read.threadpool.size` to 0 or remove it from the configuration to disable the feature. After changing these properties, restart your cluster.

The following is an example configuration for hedged reads for HBase.

```
<property>
  <name>dfs.client.hedged.read.threadpool.size</name>
  <value>20</value>  <!-- 20 threads -->
</property>
<property>
  <name>dfs.client.hedged.read.threshold.millis</name>
  <value>10</value>  <!-- 10 milliseconds -->
</property>
```

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

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,

Starting in CDH 5.4 (HBase 1.0), 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'}}
```

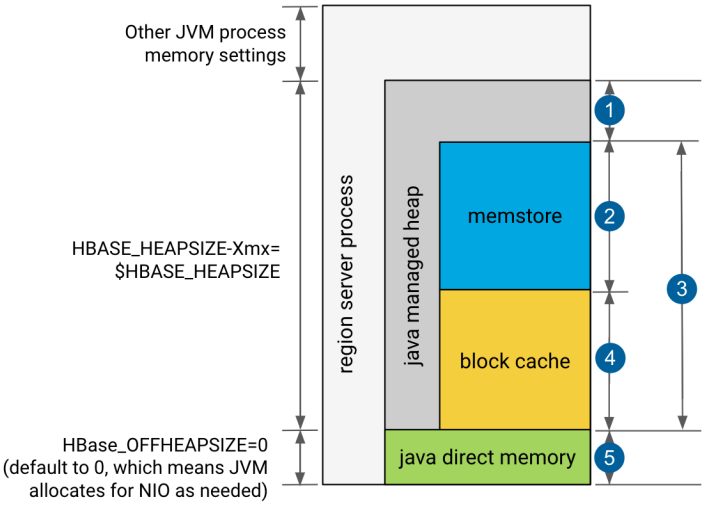
Configuring the Off-heap BucketCache

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 66 or Configuring the Off-heap BucketCache Using the Command Line on page 67. The table is followed by three diagrams that show the impacts of different blockcache settings.

Table 2: BucketCache Configuration Properties

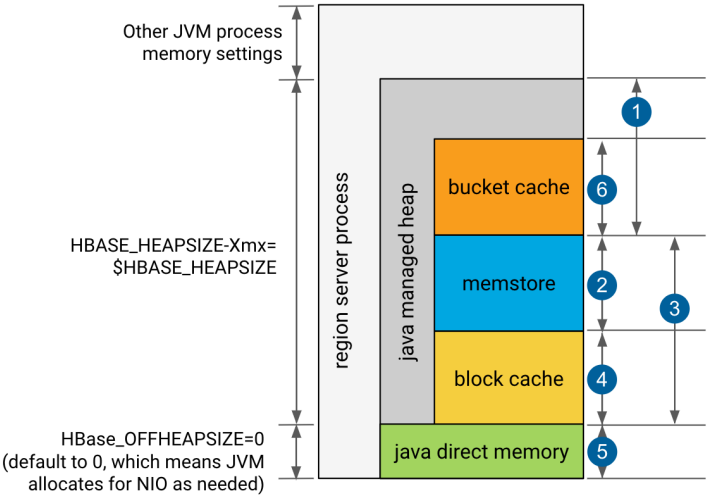
Property	Default	Description
hbase.bucketcache.combinedcache.enabled	`true`	When BucketCache is enabled, use it as a L2 cache for LruBlockCache. If set to true, indexes and Bloom filters are kept in the LruBlockCache and the data blocks are kept in the BucketCache.
`hbase.bucketcache.ioengine`	none (BucketCache is disabled by default)	Where to store the contents of the BucketCache. Its value can be `offheap`, `heap` or `file:PATH` where PATH is the path to the file that host the file-based cache.

Property	Default	Description
`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. • If less than 1.0, it represents a percentage of total heap memory size to give to the cache. • 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 to the JVM. It is automatically calculated and configured based on the following 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`

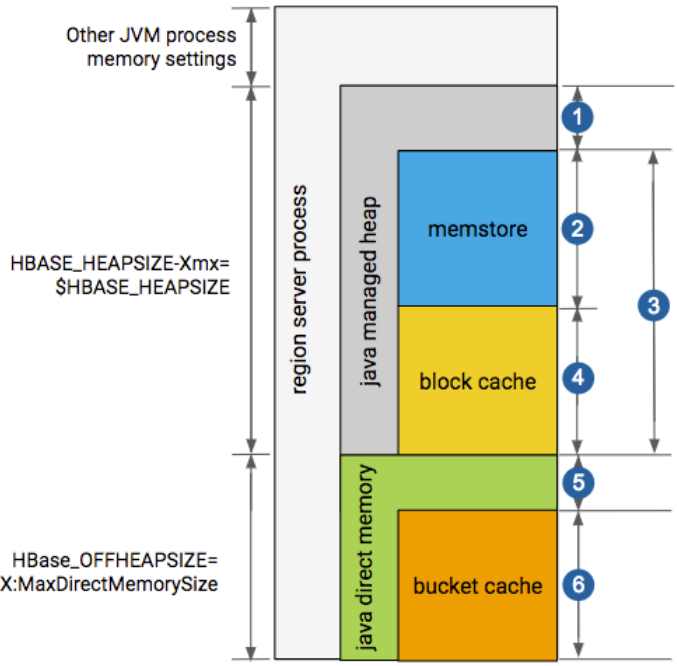


1. 20% minimum reserved for operations and rpc call queues
2. `hbase.regionserver.global.memstore.size`: default is 0.4, which means 40%
3. `hbase.regionserver.global.memstore.size` + `hfile.block.cache.size` ≤0.80, which means 80%
4. `hfile.block.cache.size`: default is 0.4, which means 40%
5. slack 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=heap`



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.

Figure 3: Default LRUCache, L1 only block cache `hbase.bucketcache.ioengine=offheap`

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. Ensure that **Enable Combined BucketCache** is selected.
   b. Set **BucketCache IOEngine** to `offheap`.
   c. Update the value of **BucketCache Size** according to the required BucketCache size.
4. In the **HBase Region Server Environment Advanced Configuration Snippet for hbase-env.sh**, edit the following parameters:
   - `HBASE_OFFHEAPSIZE`: Set it to a value (such as 5G) that accommodates your required L2 cache size, in addition to space reserved for cache management.

- HBASE_OPTS: Add the JVM option `--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`.

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. 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 5.14.x. See Cloudera Documentation for information specific to other releases.

1. Verify the RegionServer's off-heap size, and if necessary, tune it by editing the `hbase-env.sh` file and adding a line like the following:

```
HBASE_OFFHEAPSIZE=5G
```

   Set it to a value that accommodates your BucketCache size + the additional space required for DFSClient short circuit reads (default is 1 GB). For example, if the BucketCache size is 8 GB, set `HBASE_OFFHEAPSIZE=9G`.

2. Configure the MaxDirectMemorySize option for the RegionServers JVMS. This can be done adding the following line in `hbase-env.sh`:

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

   Replace <size> with the same value set for `HBASE_OFFHEAPSIZE`.

3. Next, in the `hbase-site.xml` files on the RegionServers, configure the properties in Table 2: BucketCache Configuration Properties on page 63 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>
```

4. 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 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. Click **Save Changes** to commit the changes.
7. Restart the role.
8. Restart the service.

Configure the Scanner Heartbeat Using the Command Line

1. Edit `hbase-site.xml` and add the following properties, modifying the values as needed.

```
<property>
  <name>hbase.rpc.timeout</name>
  <value>60000</value>
</property>
<property>
  <name>hbase.client.scanner.timeout.period</name>
  <value>60000</value>
</property>
<property>
  <name>hbase.cells.scanned.per.heartbeat.check</name>
```

```
  <value>10000</value>
</property>
```

2. Distribute the modified `hbase-site.xml` to all your cluster hosts and restart the HBase master and RegionServer processes for the change to take effect.

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.compactions.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 + (higer - lower) * compactionPressure
```

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

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 into the field and modify the values as needed. See Configure the Compaction Speed Using the Command Line on page 69 for an explanation of the properties.
5. Click **Save Changes** to commit the changes.
6. Restart the role.
7. Restart the service.

Configure the Compaction Speed Using the Command Line

1. Edit `hbase-site.xml` and add the relevant properties, modifying the values as needed. Default values are shown. `hbase.offpeak.start.hour` and `hbase.offpeak.end.hour` have no default values; this configuration sets the off-peak hours from 20:00 (8 PM) to 6:00 (6 AM).

```
<property>
  <name>hbase.regionserver.throughput.controller</name>

<value>org.apache.hadoop.hbase.regionserver.compactions.PressureAwareCompactionThroughputController</value>
</property>
<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>
  <value>When to begin using off-peak compaction settings, expressed as an integer
between 0 and 23.</value>
</property>
<property>
  <name>hbase.offpeak.end.hour</name>
  <value>6</value>
  <value>When to stop using off-peak compaction settings, expressed as an integer between
 0 and 23.</value>
</property>
<property>
  <name>hbase.hstore.compaction.throughput.tune.period</name>
  <value>60000</value>
  <description>
</property>
```

2. Distribute the modified `hbase-site.xml` to all your cluster hosts and restart the HBase master and RegionServer processes for the change to take effect.

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 72 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 Using the Command Line

Important:

- Follow these command-line instructions on systems that do not use Cloudera Manager.
- This information applies specifically to CDH 5.14.x. See Cloudera Documentation for information specific to other releases.

To enable hedged reads for HBase, edit the `hbase-site.xml` file on each server. Set `dfs.client.hedged.read.threadpool.size` to the number of threads to dedicate to running hedged threads, and set the `dfs.client.hedged.read.threshold.millis` configuration property to the number of milliseconds to wait before starting a second read against a different block replica. Set `dfs.client.hedged.read.threadpool.size` to 0 or remove it from the configuration to disable the feature. After changing these properties, restart your cluster.

The following is an example configuration for hedged reads for HBase.

```
<property>
  <name>dfs.client.hedged.read.threadpool.size</name>
  <value>20</value>  <!-- 20 threads -->
</property>
<property>
  <name>dfs.client.hedged.read.threshold.millis</name>
  <value>10</value>  <!-- 10 milliseconds -->
</property>
```

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 70.

Dynamically Loading a Custom Filter

CDH 5.5 and higher adds (and enables by default) 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);
  }

}
```

Writing Data to HBase

To write data to HBase, you use methods of the `HTableInterface` 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 in CDH 5, and 3 in previous versions. 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 HTableInterface 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

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
```

In CDH 5, 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.halxg.cloudera.com/10.20.188.121:8032
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.halxg.cloudera.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.halxg.cloudera.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.

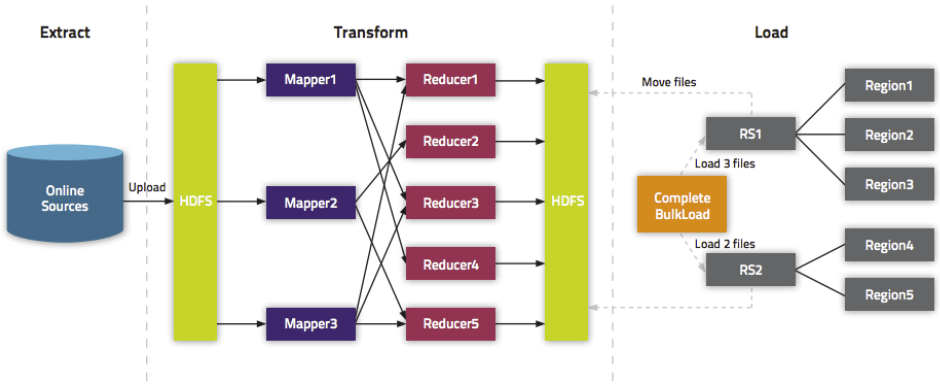
Note: Because they bypass the WAL, BulkLoad operations are not propagated between clusters using replication. If you need the data on all replicated clusters, you must perform the BulkLoad on each cluster.

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. This functionality is available in CDH 5.5 and higher.

```
<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

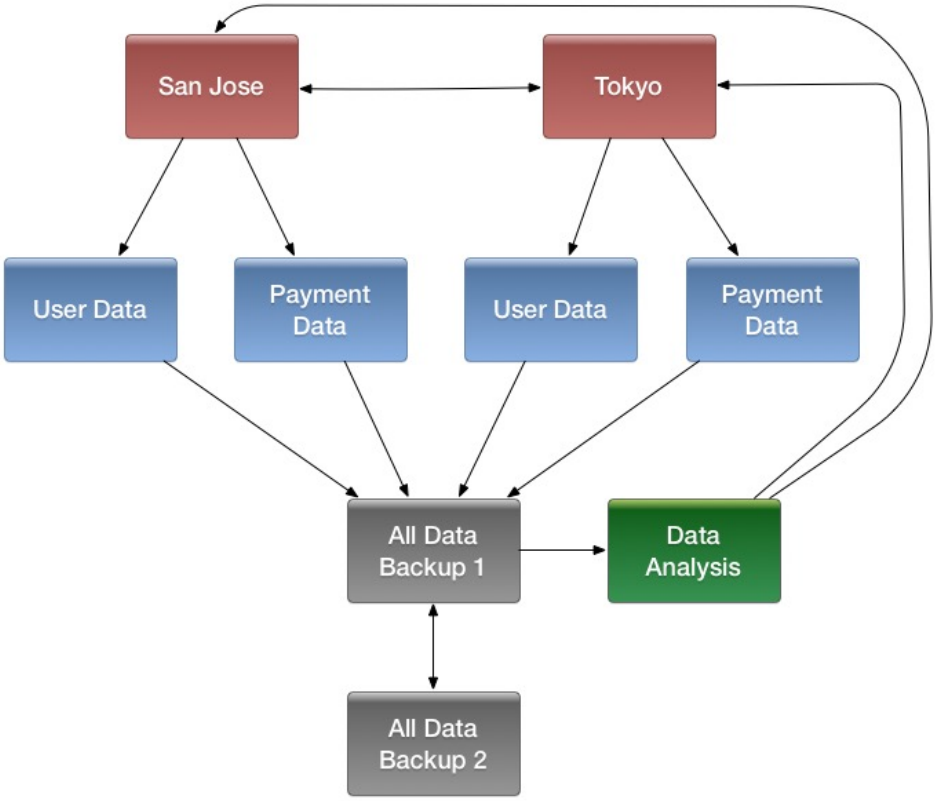
If your data is already in an HBase cluster, replication is useful for getting the data into additional HBase clusters. In HBase, cluster replication refers to keeping one cluster state synchronized with that of another cluster, using the write-ahead log (WAL) of the source cluster to propagate the changes. 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. In CDH 5, 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 on page 117 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 on page 123.
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 10.10.1.49:62020
```

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 on page 123 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. See Installing and Starting the HBase Thrift Server on page 26. 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 iexample 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 the HBase REST Server 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 two different *sinks* designed to work with HBase: HBaseSink (org.apache.flume.sink.hbase.HBaseSink) and AsyncHBaseSink (org.apache.flume.sink.hbase.AsyncHBaseSink). HBaseSink supports HBase IPC calls introduced in HBase 0.96, and allows you to write data to an HBase cluster that is secured by Kerberos, whereas AsyncHBaseSink does not. However, AsyncHBaseSink uses an asynchronous model and guarantees atomicity at the row level.

You configure HBaseSink and AsyncHBaseSink nearly identically. Following is an example configuration for each. Bold lines highlight differences in the configurations. For full documentation about configuring HBaseSink and AsyncHBaseSink, 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 either AsyncHBaseEventSerializer or HBaseEventSerializer. The HBaseEventSerializer converts Flume Events into one or more HBase Puts, sends them to the HBase cluster, and is closed when the HBaseSink stops. AsyncHBaseEventSerializer starts and listens for Events. When it receives an Event, it calls the setEvent method and then calls the getActions and getIncrements methods. When the AsyncHBaseSink is stopped, the serializer cleanUp method is called. These methods return PutRequest and AtomicIncrementRequest, which are part of the asynchbase API.

AsyncHBaseSink:

```
#Use the AsyncHBaseSink
host1.sinks.sink1.type = org.apache.flume.sink.hbase.AsyncHBaseSink
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 SimpleAsyncHbaseEventSerializer that comes with Flume
host1.sinks.sink1.serializer = org.apache.flume.sink.hbase.SimpleAsyncHbaseEventSerializer
host1.sinks.sink1.serializer.incrementColumn = icol
host1.channels.ch1.type=memory
```

HBaseSink:

```
#Use the HBaseSink
host1.sinks.sink1.type = org.apache.flume.sink.hbase.HBaseSink
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 SimpleHbaseEventSerializer that comes with Flume
host1.sinks.sink1.serializer = org.apache.flume.sink.hbase.SimpleHbaseEventSerializer
```

```
host1.sinks.sink1.serializer.incrementColumn = icol
host1.channels.ch1.type=memory
```

The following serializer, taken from 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 row is defined in the event header. When each event is received, a counter is incremented to track the number of events received.

```
   /**
  * A serializer for the AsyncHBaseSink, 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 AsyncHbaseEventSerializer {
   private byte[] table;
   private byte[] colFam;
   private Event currentEvent;
   private byte[][] columnNames;
   private final List<PutRequest> puts = new ArrayList<PutRequest>();
   private final List<AtomicIncrementRequest> incs = new
ArrayList<AtomicIncrementRequest>();
   private byte[] currentRowKey;
   private final byte[] eventCountCol = "eventCount".getBytes();

   @Override
   public void initialize(byte[] table, byte[] cf) {
     this.table = table;
     this.colFam = cf;
   }

   @Override
   public void setEvent(Event event) {
     // Set the event and verify that the rowKey is not present
     this.currentEvent = event;
     String rowKeyStr = currentEvent.getHeaders().get("rowKey");
     if (rowKeyStr == null) {
       throw new FlumeException("No row key found in headers!");
     }
     currentRowKey = rowKeyStr.getBytes();
   }

   @Override
   public List<PutRequest> 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 PutRequest for each column.
       PutRequest req = new PutRequest(table, currentRowKey, colFam,
               columnNames[i], cols[i].getBytes());
       puts.add(req);
     }
     return puts;
   }

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

   @Override
   public void cleanUp() {
     table = null;
     colFam = null;
     currentEvent = null;
     columnNames = null;
     currentRowKey = null;
```

```
    }

    @Override
    public void configure(Context context) {
      //Get the column names from the configuration
      String cols = new String(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 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`.

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**.
8. To configure Kerberos authentication between REST clients and servers, see Configure Authentication for the HBase REST and Thrift Gateways.

Installing the REST Server Using the Command Line

Important:

- Follow these command-line instructions on systems that do not use Cloudera Manager.
- This information applies specifically to CDH 5.14.x. See Cloudera Documentation for information specific to other releases.

Follow these instructions for each HBase host fulfilling the REST server role.

- To start the REST server as a foreground process, use the following command:

```
$ bin/hbase rest start
```

- To start the REST server as a background process, use the following command:

```
$ bin/hbase-daemon.sh start rest
```

- To use a different port than the default of 8080, use the `-p` option.
- To stop a running HBase REST server, use the following command:

```
$ bin/hbase-daemon.sh stop rest
```

- To configure Kerberos authentication between REST clients and servers, see Configure Authentication for the HBase REST and Thrift Gateways.

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 3: 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 4: 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 5: 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 6: 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: `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:
			`curl -vi -X GET \ -H "Accept: text/xml" \ "http://example.com:20550/users/row1/cf:a"`
			Specific version:
			`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 7: 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:
		`<Scanner batch="100">`
		`  <filter>`
		`  {`
		`  "type": "PrefixFilter",`
		`  "value": "u123"`
		`  }`
		`  </filter>`
		`</Scanner>`
		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/145869072824375522207"`
/*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/145869072824375522207"`

Table 8: 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:
			`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"`
			JSON:
			`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. Click **Save Changes** to commit the changes.
8. Restart the RegionServer roles.

Configuring MultiWAL Support Using the Command Line

Important:

- Follow these command-line instructions on systems that do not use Cloudera Manager.
- This information applies specifically to CDH 5.14.x. See Cloudera Documentation for information specific to other releases.

1. Edit `hbase-site.xml` on each RegionServer where you want to enable MultiWAL. Add the following property by pasting the XML.

```
<property>
  <name>hbase.wal.provider</name>
  <value>multiwal</value>
</property>
```

2. Stop and restart the RegionServer.

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 5.4 and higher 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.

Enabling HFile Version 3 Using the Command Line

Important:

- Follow these command-line instructions on systems that do not use Cloudera Manager.
- This information applies specifically to CDH 5.14.x. See Cloudera Documentation for information specific to other releases.

Paste the following XML into `hbase-site.xml`.

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

Restart HBase. Changes will take effect for a given region during its 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 9: 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 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 9: HBase MOB Cache Properties on page 107 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.

Configuring the MOB Cache Using the Command Line

Important:

- Follow these command-line instructions on systems that do not use Cloudera Manager.
- This information applies specifically to CDH 5.14.x. See Cloudera Documentation for information specific to other releases.

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.

To customize the configuration of the MOB file reader's cache on each RegionServer, configure the MOB cache properties in the RegionServer's `hbase-site.xml`. Customize the configuration to suit your environment, and restart or rolling restart the RegionServer. Cloudera recommends testing your configuration with the default settings first. The following example sets the `hbase.mob.cache.evict.period` property to 5000 seconds. See Table 9: HBase MOB Cache Properties on page 107 for a full list of configurable properties for HBase MOB.

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

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 HBase Shell commands `compact_mob` and `major_compact_mob`. Each of these commands requires the first parameter to be the table name, and takes an optional column family name as the second argument. 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_mob 't1'
hbase> compact_mob 't1', 'f1'
hbase> major_compact_mob 't1'
hbase> major_compact_mob 't1', 'f1'
```

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 5.14.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.

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. Click **Save Changes** to commit the changes.
8. Restart the role.
9. Restart the service.

Expose HBase Metrics to Ganglia Using the Command Line

Important:

- Follow these command-line instructions on systems that do not use Cloudera Manager.
- This information applies specifically to CDH 5.14.x. See Cloudera Documentation for information specific to other releases.

1. Edit `/etc/hbase/conf/hadoop-metrics2-hbase.properties` on the master or RegionServers you want to monitor, and add the following properties, 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
```

2. Restart the master or RegionServer.

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 Configuring Hadoop Security in CDH 5 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 5 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 10: 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.

Enable HBase Authorization Using Cloudera Manager

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**.

Enable HBase Authorization Using the Command Line

Important:

- Follow these command-line instructions on systems that do not use Cloudera Manager.
- This information applies specifically to CDH 5.14.x. See Cloudera Documentation for information specific to other releases.

To enable HBase authorization, add the following properties to the `hbase-site.xml` file *on every HBase server host (Master or RegionServer)*:

```
<property>
     <name>hbase.security.authorization</name>
     <value>true</value>
</property>
<property>
  <name>hbase.security.exec.permission.checks</name>
  <value>true</value>
</property>
<property>
     <name>hbase.coprocessor.master.classes</name>
     <value>org.apache.hadoop.hbase.security.access.AccessController</value>
</property>
<property>
     <name>hbase.coprocessor.region.classes</name>

<value>org.apache.hadoop.hbase.security.token.TokenProvider,org.apache.hadoop.hbase.security.access.AccessController</value>
</property>
```

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'))" }
```

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 86
- Configuring Secure HBase Replication

HBase Replication

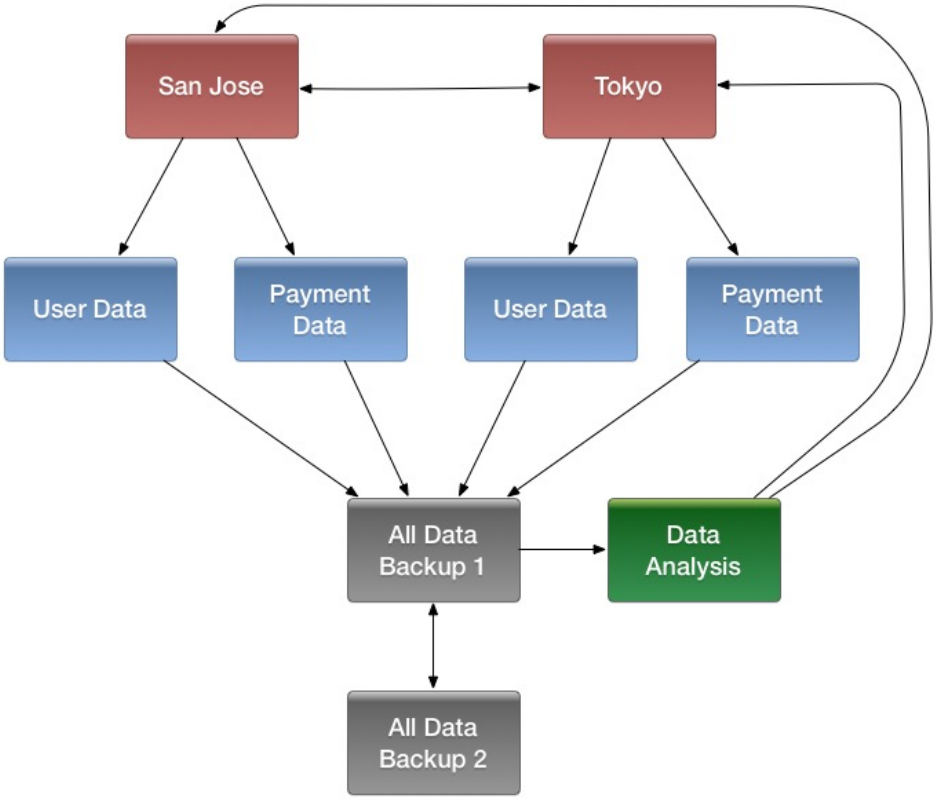
If your data is already in an HBase cluster, replication is useful for getting the data into additional HBase clusters. In HBase, cluster replication refers to keeping one cluster state synchronized with that of another cluster, using the write-ahead log (WAL) of the source cluster to propagate the changes. 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. 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. In CDH 5, 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.

Notes about Replication

- The timestamps of the replicated HLog entries are kept intact. In case of a collision (two entries identical as to row key, column family, column qualifier, and timestamp) only the entry arriving later write will be read.
- Increment Column Values (ICVs) are treated as simple puts when they are replicated. In the case where each side of replication is active (new data originates from both sources, which then replicate each other), this may be undesirable, creating identical counters that overwrite one another. (See https://issues.apache.org/jira/browse/HBase-2804.)
- Make sure the source and destination clusters are time-synchronized with each other. Cloudera recommends you use Network Time Protocol (NTP).
- Some changes are not replicated and must be propagated through other means, such as Snapshots or CopyTable.
  - Data that existed in the active cluster before replication was enabled.
  - Operations that bypass the WAL, such as when using BulkLoad or API calls such as `writeToWal(false)`.
  - Table schema modifications.

Requirements

Before configuring replication, make sure your environment meets the following requirements:

- You must manage ZooKeeper yourself. It must not be managed by HBase, and must be available throughout the deployment.
- Each host in both clusters must be able to reach every other host, including those in the ZooKeeper cluster.
- Both clusters must be running the same major version of CDH; for example CDH 5.
- Every table that contains families that are scoped for replication must exist on each cluster and have exactly the same name. If your tables do not yet exist on the destination cluster, see Creating the Empty Table On the Destination Cluster on page 123.
- HBase version 0.92 or greater is required for complex replication topologies, such as active-active.

Deploying HBase Replication

Follow these steps to enable replication from one cluster to another.

> **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 on page 123.
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 10.10.1.49:62020
```

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.

Replicating Across Three or More Clusters

When configuring replication among three or more clusters, Cloudera recommends you enable `KEEP_DELETED_CELLS` on column families in the destination cluster, where `REPLICATION_SCOPE=1` in the source cluster. The following commands show how to enable this configuration using HBase Shell.

- On the source cluster:

```
create 't1',{NAME=>'f1', REPLICATION_SCOPE=>1}
```

- On the destination cluster:

```
create 't1',{NAME=>'f1', KEEP_DELETED_CELLS=>'true'}
```

Enabling Replication on a Specific Table

To enable replication for a specific table on the source cluster, run the `enable_table_replication <table>` command from the HBase shell on a cluster where a peer has been configured.

Running `enable_table_replication <table>` does the following:

1. Verifies that the table exists on the source cluster.
2. If the table does not exist on the remote cluster, uses the peer configuration to duplicate the table schema (including splits) on the remote cluster.
3. Enables replication on that table.

Configuring Secure Replication

The following steps describe how to set up secure replication between clusters. The steps are the same whether your clusters are all in the same realm or not, with the exception of the last step.

The last step involves setting up custom secure replication configurations per peer. This can be convenient when you need to replicate to a cluster that uses different cross-realm authentication rules than the source cluster. For example, a cluster in Realm A may be allowed to replicate to Realm B and Realm C, but Realm B may not be allowed to replicate to Realm C. If you do not need this feature, skip the last step.

To use this feature, service-level principals and keytabs (specific to HBase) must be specified when you create the cluster peers using HBase Shell.

Note: HBase peer-to-peer replication from a non-Kerberized cluster to a Kerberized cluster is not supported.

1. Set up Kerberos on your cluster, as described in Enabling Kerberos Authentication Using the Wizard.
2. If necessary, configure Kerberos cross-realm authentication.
   - At the command line, use the `list_principals` command to list the `kdc`, `admin_server`, and `default_domain` for each realm.
   - Add this information to each cluster using Cloudera Manager. For each cluster, go to **HDFS** > **Configuration** > **Trusted Kerberos Realms**. Add the target and source. This requires a restart of HDFS.
3. Configure ZooKeeper.
4. Configure the following HDFS parameters on both clusters, in Cloudera Manager or in the listed files if you do not use Cloudera Manager:

Note:

If you use Cloudera Manager to manage your cluster, do not set these properties directly in configuration files, because Cloudera Manager will overwrite or ignore these settings. You must set these properties in Cloudera Manager.

For brevity, the Cloudera Manager setting names are not listed here, but you can search by property name. For instance, in the HDFS service configuration screen, search for **dfs.encrypt.data.transfer**. The **Enable Data Transfer Encryption** setting is shown. Selecting the box is equivalent to setting the value to `true`.

```
<!-- In hdfs-site.xml or advanced configuration snippet -->
<property>
  <name>dfs.encrypt.data.transfer</name>
  <value>true</value>
</property>
<property>
  <name>dfs.data.transfer.protection</name>
  <value>privacy</value>
</property>

<!-- In core-site.xml or advanced configuration snippet -->
<property>
  <name>hadoop.security.authorization</name>
  <value>true</value>
</property>
<property>
  <name>hadoop.rpc.protection</name>
  <value>privacy</value>
</property>
<property>
  <name>hadoop.security.crypto.cipher.suite</name>
  <value>AES/CTR/NoPadding</value>
</property>
<property>
  <name>hadoop.ssl.enabled</name>
  <value>true</value>
</property>
```

5. Configure the following HBase parameters on both clusters, using Cloudera Manager or in `hbase-site.xml` if you do not use Cloudera Managert.

```
<!-- In hbase-site.xml -->
<property>
  <name>hbase.rpc.protection</name>
  <value>privacy</value>
</property>
<property>
  <name>hbase.thrift.security.qop</name>
  <value>auth-conf</value>
</property>
<property>
  <name>hbase.thrift.ssl.enabled</name>
  <value>true</value>
</property>
<property>
  <name>hbase.rest.ssl.enabled</name>
  <value>true</value>
</property>
<property>
  <name>hbase.ssl.enabled</name>
  <value>true</value>
</property>
<property>
  <name>hbase.security.authentication</name>
  <value>kerberos</value>
</property>
<property>
```

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

6. Add the cluster peers using the simplified `add_peer` syntax, as described in Add Peer.

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

7. If you need to add any peers which require custom security configuration, modify the `add_peer` syntax, using the following examples as a model.

```
add_peer 'vegas', CLUSTER_KEY => 'zk1.vegas.example.com:2181:/hbase',
        CONFIG => {'hbase.master.kerberos.principal'          => 'hbase/_HOST@TO_VEGAS',

                   'hbase.regionserver.kerberos.principal'    => 'hbase/_HOST@TO_VEGAS',

                      'hbase.regionserver.keytab.file'            =>
'/keytabs/vegas_hbase.keytab',
                      'hbase.master.keytab.file'                  =>
'/keytabs/vegas_hbase.keytab'},
          TABLE_CFS => {"tbl" => [cf1']}

add_peer 'atlanta', CLUSTER_KEY => 'zk1.vegas.example.com:2181:/hbase',
         CONFIG => {'hbase.master.kerberos.principal'           =>
'hbase/_HOST@TO_ATLANTA',
                      'hbase.regionserver.kerberos.principal'     =>
'hbase/_HOST@TO_ATLANTA',
                      'hbase.regionserver.keytab.file'            =>
'/keytabs/atlanta_hbase.keytab',
                      'hbase.master.keytab.file'                  =>
'/keytabs/atlanta_hbase.keytab'},
          TABLE_CFS => {"tbl" => [cf2']}
```

Disabling Replication at the Peer Level

Use the command `disable_peer ("<peerID>")` to disable replication for a specific peer. This will stop replication to the peer, but the logs will be kept for future reference.

> **Note:** This log accumulation is a powerful side effect of the `disable_peer` command and can be used to your advantage. See Initiating Replication When Data Already Exists on page 123.

To re-enable the peer, use the command `enable_peer(<"peerID">)`. Replication resumes.

Examples:

- To disable peer 1:

```
disable_peer("1")
```

- To re-enable peer 1:

```
enable_peer("1")
```

Stopping Replication in an Emergency

If replication is causing serious problems, you can stop it while the clusters are running.

Open the shell on the source cluster and use the `disable_peer` command for each peer, then the `disable_table_replication` command. For example:

```
hbase> disable_peer("1")
hbase> disable_table_replication
```

Already queued edits will be replicated after you use the `disable_table_replication` command, but new entries will not. See Understanding How WAL Rolling Affects Replication on page 124.

To start replication again, use the `enable_peer` command.

Creating the Empty Table On the Destination Cluster

If the table to be replicated does not yet exist on the destination cluster, you must create it. The easiest way to do this is to extract the schema using HBase Shell.

1. On the source cluster, `describe` the table using HBase Shell. The output below has been reformatted for readability.

```
hbase> describe acme_users

Table acme_users is ENABLED
acme_users
COLUMN FAMILIES DESCRIPTION
{NAME => 'user', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'NONE',
REPLICATION_SCOPE => '0', VERSIONS => '3', COMPRESSION => 'NONE',
MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE',
BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'false'}
```

2. Copy the output and make the following changes:
   - For the TTL, change `FOREVER` to `org.apache.hadoop.hbase.HConstants::FOREVER`.
   - Add the word `CREATE` before the table name.
   - Remove the line `COLUMN FAMILIES DESCRIPTION` and everything above the table name.

   The result is a command like the following:

```
"CREATE 'cme_users' ,

{NAME => 'user', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'NONE',
REPLICATION_SCOPE => '0', VERSIONS => '3', COMPRESSION => 'NONE',
MIN_VERSIONS => '0', TTL => org.apache.hadoop.hbase.HConstants::FOREVER,
KEEP_DELETED_CELLS => 'FALSE',
BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'false'}
```

3. On the destination cluster, paste the command from the previous step into HBase Shell to create the table.

Initiating Replication When Data Already Exists

You may need to start replication from some point in the past. For example, suppose you have a primary HBase cluster in one location and are setting up a disaster-recovery (DR) cluster in another. To initialize the DR cluster, you need to copy over the existing data from the primary to the DR cluster, so that when you need to switch to the DR cluster you have a full copy of the data generated by the primary cluster. Once that is done, replication of new data can proceed as normal.

One way to do this is to take advantage of the write accumulation that happens when a replication peer is disabled.

1. Start replication.
2. Add the destination cluster as a peer and immediately disable it using `disable_peer`.
3. On the source cluster, take a snapshot of the table and export it. The snapshot command flushes the table from memory for you.
4. On the destination cluster, import and restore the snapshot.

5. Run `enable_peer` to re-enable the destination cluster.

Replicating Pre-existing Data in an Active-Active Deployment

In the case of active-active replication, run the `copyTable` job before starting the replication. (If you start the job after enabling replication, the second cluster will re-send the data to the first cluster, because `copyTable` does not edit the `clusterId` in the mutation objects. The following is one way to accomplish this:

1. Run the `copyTable` job and note the start timestamp of the job.
2. Start replication.
3. Run the `copyTable` job again with a start time equal to the start time you noted in step 1.

This results in some data being pushed back and forth between the two clusters; but it minimizes the amount of data.

Understanding How WAL Rolling Affects Replication

When you add a new peer cluster, it only receives new writes from the source cluster **since the last time the WAL was rolled**.

The following diagram shows the consequences of adding and removing peer clusters with unpredictable WAL rolling occurring. Follow the time line and notice which peer clusters receive which writes. Writes that occurred before the WAL is rolled are **not** retroactively replicated to new peers that were not participating in the cluster before the WAL was rolled.

Configuring Secure HBase Replication

If you want to make HBase Replication secure, follow the instructions under HBase Authentication.

Restoring Data From A Replica

One of the main reasons for replications is to be able to restore data, whether during disaster recovery or for other reasons. During restoration, the *source* and *sink* roles are reversed. The source is the replica cluster, and the sink is the cluster that needs restoration. This can be confusing, especially if you are in the middle of a disaster recovery scenario. The following image illustrates the role reversal between normal production and disaster recovery.

Follow these instructions to recover HBase data from a replicated cluster in a disaster recovery scenario.

1. Change the value of the column family property `REPLICATION_SCOPE` on the sink to `0` for each column to be restored, so that its data will not be replicated during the restore operation.
2. Change the value of the column family property `REPLICATION_SCOPE` on the source to `1` for each column to be restored, so that its data will be replicated.
3. Use the `CopyTable` or `distcp` commands to import the data from the backup to the sink cluster, as outlined in Initiating Replication When Data Already Exists on page 123.
4. Add the sink as a replication peer to the source, using the `add_peer` command as discussed in Deploying HBase Replication on page 118. If you used `distcp` in the previous step, restart or rolling restart both clusters, so that the RegionServers will pick up the new files. If you used `CopyTable`, you do not need to restart the clusters. New data will be replicated as it is written.
5. When restoration is complete, change the `REPLICATION_SCOPE` values back to their values before initiating the restoration.

Verifying that Replication is Working

To verify that HBase replication is working, follow these steps to confirm data has been replicated from a source cluster to a remote destination cluster.

1. Install and configure YARN on the source cluster.

   If YARN cannot be used in the source cluster, configure YARN on the destination cluster to verify replication. If neither the source nor the destination clusters can have YARN installed, you can configure the tool to use local mode; however, performance and consistency could be negatively impacted.

2. Make sure that you have the required permissions:

   - You have sudo permissions to run commands as the hbase user, or a user with admin permissions on both clusters.
   - You are an hbase user configured for submitting jobs with YARN.

> **Note:** To use the hbase user in a secure cluster, use Cloudera Manager to add the hbase user as a YARN whitelisted user. If you are running Cloudera Manager 5.8 or higher, and are running a new installation, the hbase user is already added to the whitelisted users. In addition, /user/hbase should exist on HDFS and owned as the hbase user, because YARN will create a job staging directory there.

3. Run the `VerifyReplication` command:

```
src-node$ sudo -u hbase hbase
org.apache.hadoop.hbase.mapreduce.replication.VerifyReplication peer1 table1
...
       org.apache.hadoop.hbase.mapreduce.replication.VerifyReplication$Verifier$Counters

                BADROWS=2
                CONTENT_DIFFERENT_ROWS=1
                GOODROWS=1
                ONLY_IN_PEER_TABLE_ROWS=1
        File Input Format Counters
                Bytes Read=0
        File Output Format Counters
                Bytes Written=0
```

The following table describes the `VerifyReplication` counters:

Table 11: VerifyReplication Counters

Counter	Description
`GOODROWS`	Number of rows. On both clusters, and all values are the same.
`CONTENT_DIFFERENT_ROWS`	The key is the same on both source and destination clusters for a row, but the value differs.
`ONLY_IN_SOURCE_TABLE_ROWS`	Rows that are only present in the source cluster but not in the destination cluster.
`ONLY_IN_PEER_TABLE_ROWS`	Rows that are only present in the destination cluster but not in the source cluster.
`BADROWS`	Total number of rows that differ from the source and destination clusters; the sum of `CONTENT_DIFFERENT_ROWS` + `ONLY_IN_SOURCE_TABLE_ROWS` + `ONLY_IN_PEER_TABLE_ROWS`

By default, `VerifyReplication` compares the entire content of `table1` on the source cluster against `table1` on the destination cluster that is configured to use the replication peer `peer1`.

Use the following options to define the period of time, versions, or column families

Table 12: VerifyReplication Counters

Option	Description
`--starttime=<timestamp>`	Beginning of the time range, in milliseconds. Time range is forever if no end time is defined.
`--endtime=<timestamp>`	End of the time range, in milliseconds.
`--versions=<versions>`	Number of cell versions to verify.
`--families=<cf1,cf2,..>`	Families to copy; separated by commas.

The following example, verifies replication only for rows with a timestamp range of one day:

```
src-node$ sudo -u hbase hbase
org.apache.hadoop.hbase.mapreduce.replication.VerifyReplication --starttime=1472499077000
 --endtime=1472585477000 --families=c1  peer1 table1
```

Replication Caveats

- Two variables govern replication: `hbase.replication` as described above under Deploying HBase Replication on page 118, and a replication znode. Stopping replication (using `disable_table_replication` as above) sets the znode to `false`. Two problems can result:
  - If you add a new RegionServer to the active cluster while replication is stopped, its current log will not be added to the replication queue, because the replication znode is still set to `false`. If you restart replication at this point (using `enable_peer`), entries in the log will not be replicated.
  - Similarly, if a log rolls on an existing RegionServer on the active cluster while replication is stopped, the new log will not be replicated, because the replication znode was set to `false` when the new log was created.
- In the case of a long-running, write-intensive workload, the destination cluster may become unresponsive if its meta-handlers are blocked while performing the replication. CDH 5 provides three properties to deal with this problem:
  - `hbase.regionserver.replication.handler.count` - the number of replication handlers in the destination cluster (default is 3). Replication is now handled by separate handlers in the destination cluster to avoid the above-mentioned sluggishness. Increase it to a high value if the ratio of active to passive RegionServers is high.
  - `replication.sink.client.retries.number` - the number of times the HBase replication client at the sink cluster should retry writing the WAL entries (default is 1).
  - `replication.sink.client.ops.timeout` - the timeout for the HBase replication client at the sink cluster (default is 20 seconds).
- For namespaces, tables, column families, or cells with associated ACLs, the ACLs themselves are not replicated. The ACLs need to be re-created manually on the target table. This behavior opens up the possibility for the ACLs could be different in the source and destination cluster.

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.

Enabling HBase High Availability Using Cloudera Manager

1. Go to the HBase service.
2. Follow the process for adding a role instance and add a backup Master to a different host than the one on which the active Master is running.

Enabling HBase High Availability Using the Command Line

To configure backup Masters, create a new file in the `conf/` directory which will be distributed across your cluster, called `backup-masters`. For each backup Master you want to start, add a new line with the hostname where the Master should be started. Each host that will run a Master needs to have all of the configuration files available. In general, it is a good practice to distribute the entire `conf/` directory across all cluster nodes.

After saving and distributing the file, restart your cluster for the changes to take effect. When the master starts the backup Masters, messages are logged. In addition, you can verify that an `HMaster` process is listed in the output of the `jps` command on the nodes where the backup Master should be running.

```
$ jps
 15930 HRegionServer
 16194 Jps
 15838 HQuorumPeer
 16010 HMaster
```

To stop a backup Master without stopping the entire cluster, first find its process ID using the `jps` command, then issue the `kill` command against its process ID.

```
$ kill 16010
```

HBase Read Replicas

CDH 5.4 introduces *read replicas*. Without read replicas, only one RegionServer services a read request from a client, regardless of whether RegionServers are colocated with other DataNodes that have local access to the same block. This ensures consistency of the data being read. However, a RegionServer can become a bottleneck due to an underperforming RegionServer, network problems, or other reasons that could cause slow reads.

With read replicas enabled, the HMaster distributes read-only copies of regions (*replicas*) to different RegionServers in the cluster. One RegionServer services the default or *primary* replica, which is the only replica which can service write requests. If the RegionServer servicing the primary replica is down, writes will fail.

Other RegionServers serve the *secondary* replicas, follow the primary RegionServer and only see committed updates. The secondary replicas are read-only, and are unable to service write requests. The secondary replicas can be kept up to date by reading the primary replica's HFiles at a set interval or by replication. If they use the first approach, the secondary replicas may not reflect the most recent updates to the data when updates are made and the RegionServer has not yet flushed the memstore to HDFS. If the client receives the read response from a secondary replica, this is indicated by marking the read as "stale". Clients can detect whether or not the read result is stale and react accordingly.

Replicas are placed on different RegionServers, and on different racks when possible. This provides a measure of high availability (HA), as far as reads are concerned. If a RegionServer becomes unavailable, the regions it was serving can

still be accessed by clients even before the region is taken over by a different RegionServer, using one of the secondary replicas. The reads may be stale until the entire WAL is processed by the new RegionServer for a given region.

For any given read request, a client can request a faster result even if it comes from a secondary replica, or if consistency is more important than speed, it can ensure that its request is serviced by the primary RegionServer. This allows you to decide the relative importance of consistency and availability, in terms of the CAP Theorem, in the context of your application, or individual aspects of your application, using Timeline Consistency semantics.

Timeline Consistency

Timeline Consistency is a consistency model which allows for a more flexible standard of consistency than the default HBase model of *strong consistency*. A client can indicate the level of consistency it requires for a given read (Get or Scan) operation. The default consistency level is STRONG, meaning that the read request is only sent to the RegionServer servicing the region. This is the same behavior as when read replicas are not used. The other possibility, `TIMELINE`, sends the request to all RegionServers with replicas, including the primary. The client accepts the first response, which includes whether it came from the primary or a secondary RegionServer. If it came from a secondary, the client can choose to verify the read later or not to treat it as definitive.

Keeping Replicas Current

The read replica feature includes two different mechanisms for keeping replicas up to date:

Using a Timer

In this mode, replicas are refreshed at a time interval controlled by the configuration option `hbase.regionserver.storefile.refresh.period`. Using a timer is supported in CDH 5.4 and higher.

Using Replication

In this mode, replicas are kept current between a source and sink cluster using HBase replication. This can potentially allow for faster synchronization than using a timer. Each time a flush occurs on the source cluster, a notification is pushed to the sink clusters for the table. To use replication to keep replicas current, you must first set the column family attribute REGION_MEMSTORE_REPLICATION to `false`, then set the HBase configuration property `hbase.region.replica.replication.enabled` to `true`.

Important: Read-replica updates using replication are not supported for the `hbase:meta` table. Columns of hbase:meta must always have their REGION_MEMSTORE_REPLICATION attribute set to `false`.

Enabling Read Replica Support

Important:

Before you enable read-replica support, make sure to account for their increased heap memory requirements. Although no additional copies of HFile data are created, read-only replicas regions have the same memory footprint as normal regions and need to be considered when calculating the amount of increased heap memory required. For example, if your table requires 8 GB of heap memory, when you enable three replicas, you need about 24 GB of heap memory.

To enable support for read replicas in HBase, you must set several properties.

Table 13: HBase Read Replica Properties

Property Name	Default Value	Description
`hbase.region.replica.replication.enabled`	`false`	The mechanism for refreshing the secondary replicas. If set to `false`, secondary replicas are not guaranteed to be consistent at the row level. Secondary

Property Name	Default Value	Description
		replicas are refreshed at intervals controlled by a timer (`hbase.regionserver.storefile.refresh.period`), and so are guaranteed to be at most that interval of milliseconds behind the primary RegionServer. Secondary replicas read from the HFile in HDFS, and have no access to writes that have not been flushed to the HFile by the primary RegionServer. If `true`, replicas are kept up to date using replication. and the column family has the attribute `REGION_MEMSTORE_REPLICATION` set to `false`, Using replication for read replication of `hbase:meta` is not supported, and `REGION_MEMSTORE_REPLICATION` must always be set to `false` on the column family.
`hbase.regionserver.storefile.refresh.period`	`0` (disabled)	The period, in milliseconds, for refreshing the store files for the secondary replicas. The default value of `0` indicates that the feature is disabled. Secondary replicas update their store files from the primary RegionServer at this interval. If refreshes occur too often, this can create a burden for the NameNode. If refreshes occur too infrequently, secondary replicas will be less consistent with the primary RegionServer.

`hbase.master.loadbalancer.class`	`org.apache.hadoop.hbase.master.`
	
	`balancer.StochasticLoadBalancer`
	
	(the class name is split for formatting purposes)
		The Java class used for balancing the load of all HBase clients. The default implementation is the `StochasticLoadBalancer`, which is the only load balancer that supports reading data from secondary RegionServers.

`hbase.ipc.client.allowsInterrupt`	`true`	Whether or not to enable interruption of RPC threads at the client. The default value of `true` enables primary RegionServers to access data from other regions' secondary replicas.
`hbase.client.primaryCallTimeout.get`	`10` ms	The timeout period, in milliseconds, an HBase client's will wait for a response before the read is submitted to a secondary replica if the read request allows timeline consistency. The default value is 10. Lower values increase the number of remote procedure calls while lowering latency.
`hbase.client.primaryCallTimeout.multiget`	`10` ms	The timeout period, in milliseconds, before an HBase client's multi-get request, such as

Property Name	Default Value	Description
		`HTable.get(List<GET>)`, is submitted to a secondary replica if the multi-get request allows timeline consistency. Lower values increase the number of remote procedure calls while lowering latency.

Configure Read Replicas Using Cloudera Manager

1. Before you can use replication to keep replicas current, you must set the column attribute `REGION_MEMSTORE_REPLICATION` to `false` for the HBase table, using HBase Shell or the client API. See Activating Read Replicas On a Table on page 132.
2. Select **Clusters** > ***HBase***.
3. Click the **Configuration** tab.
4. Select **Scope** > **HBase or HBase Service-Wide**.
5. Select **Category** > **Advanced**.
6. 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.
7. Using the same XML syntax as Configure Read Replicas Using the Command Line on page 131 and the chart above, create a configuration and paste it into the text field.
8. Click **Save Changes** to commit the changes.

Configure Read Replicas Using the Command Line

Important:

- Follow these command-line instructions on systems that do not use Cloudera Manager.
- This information applies specifically to CDH 5.14.x. See Cloudera Documentation for information specific to other releases.

1. Before you can use replication to keep replicas current, you must set the column attribute `REGION_MEMSTORE_REPLICATION` to `false` for the HBase table, using HBase Shell or the client API. See Activating Read Replicas On a Table on page 132.
2. Add the properties from Table 13: HBase Read Replica Properties on page 129 to `hbase-site.xml` on each RegionServer in your cluster, and configure each of them to a value appropriate to your needs. The following example configuration shows the syntax.

```
<property>
  <name>hbase.regionserver.storefile.refresh.period</name>
  <value>0</value>
</property>
<property>
  <name>hbase.ipc.client.allowsInterrupt</name>
  <value>true</value>
  <description>Whether to enable interruption of RPC threads at the client. The default
 value of true is
    required to enable Primary RegionServers to access other RegionServers in secondary
 mode. </description>
</property>
<property>
  <name>hbase.client.primaryCallTimeout.get</name>
  <value>10</value>
</property>
<property>
  <name>hbase.client.primaryCallTimeout.multiget</name>
  <value>10</value>
</property>
```

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

Configuring Rack Awareness for Read Replicas

Rack awareness for read replicas is modeled after the mechanism used for rack awareness in Hadoop. Its purpose is to ensure that some replicas are on a different rack than the RegionServer servicing the table. The default implementation, which you can override by setting `hbase.util.ip.to.rack.determiner`, to custom implementation, is `ScriptBasedMapping`, which uses a *topology map* and a *topology script* to enforce distribution of the replicas across racks.

Creating a Topology Map

The topology map assigns hosts to racks. It is read by the topology script. A rack is a logical grouping, and does not necessarily correspond to physical hardware or location. Racks can be nested. If a host is not in the topology map, it is assumed to be a member of the default rack. The following map uses a nested structure, with two data centers which each have two racks. All services on a host that are rack-aware will be affected by the rack settings for the host.

If you use Cloudera Manager, do not create the map manually. Instead, go to **Hosts**, select the hosts to assign to a rack, and select **Actions for Selected** > **Assign Rack**.

```
<topology>
  <node name="host1.example.com" rack="/dc1/r1"/>
  <node name="host2.example.com" rack="/dc1/r1"/>
  <node name="host3.example.com" rack="/dc1/r2"/>
  <node name="host4.example.com" rack="/dc1/r2"/>
  <node name="host5.example.com" rack="/dc2/r1"/>
  <node name="host6.example.com" rack="/dc2/r1"/>
  <node name="host7.example.com" rack="/dc2/r2"/>
  <node name="host8.example.com" rack="/dc2/r2"/>
</topology>
```

Creating a Topology Script

The topology script determines rack topology using the topology map. By default, CDH uses `/etc/hadoop/conf.cloudera.YARN-1/topology.py` To use a different script, set `net.topology.script.file.name` to the absolute path of the topology script.

Activating Read Replicas On a Table

After enabling read replica support on your RegionServers, configure the tables for which you want read replicas to be created. Keep in mind that each replica increases the amount of storage used by HBase in HDFS.

At Table Creation

To create a new table with read replication capabilities enabled, set the REGION_REPLICATION property on the table. Use a command like the following, in HBase Shell:

```
hbase> create 'myTable', 'myCF', {REGION_REPLICATION => '3'}
```

By Altering an Existing Table

You can also alter an existing column family to enable or change the number of read replicas it propagates, using a command similar to the following. The change will take effect at the next major compaction.

```
hbase> disable 'myTable'
hbase> alter 'myTable', 'myCF', {REGION_REPLICATION => '3'}
hbase> enable 'myTable'
```

Requesting a Timeline-Consistent Read

To request a timeline-consistent read in your application, use the `get.setConsistency(Consistency.TIMELINE)` method before performing the `Get` or `Scan` operation.

To check whether the result is stale (comes from a secondary replica), use the `isStale()` method of the result object. Use the following examples for reference.

Get Request

```
Get get = new Get(key);
get.setConsistency(Consistency.TIMELINE);
Result result = table.get(get);
```

Scan Request

```
Scan scan = new Scan();
scan.setConsistency(CONSISTENCY.TIMELINE);
ResultScanner scanner = table.getScanner(scan);
Result result = scanner.next();
```

Scan Request to a Specific Replica

This example overrides the normal behavior of sending the read request to all known replicas, and only sends it to the replica specified by ID.

```
Scan scan = new Scan();
scan.setConsistency(CONSISTENCY.TIMELINE);
scan.setReplicaId(2);
ResultScanner scanner = table.getScanner(scan);
Result result = scanner.next();
```

Detecting a Stale Result

```
Result result = table.get(get);
if (result.isStale()) {
   ...
}
```

Getting and Scanning Using HBase Shell

You can also request timeline consistency using HBase Shell, allowing the result to come from a secondary replica.

```
hbase> get 'myTable', 'myRow', {CONSISTENCY => "TIMELINE"}

hbase> scan 'myTable', {CONSISTENCY => 'TIMELINE'}
```

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 14: 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.

HiveServer2 Security Configuration

HiveServer2 supports authentication of the Thrift client using the following methods:

- Kerberos authentication
- LDAP authentication

  Starting with CDH 5.7, clusters running LDAP-enabled HiveServer2 deployments also accept Kerberos authentication. This ensures that users are not forced to enter usernames/passwords manually, and are able to take advantage

of the multiple authentication schemes SASL offers. In CDH 5.6 and lower, HiveServer2 stops accepting delegation tokens when any alternate authentication is enabled.

Kerberos authentication is supported between the Thrift client and HiveServer2, and between HiveServer2 and secure HDFS. LDAP authentication is supported only between the Thrift client and HiveServer2.

To configure HiveServer2 to use one of these authentication modes, configure the `hive.server2.authentication` configuration property.

Note: To configure dual authentication (both Kerberos and LDAP), configure Kerberos for HiveServer2 and set the `hive.server2.authentication` property to `LDAP`.

Enabling Kerberos Authentication for HiveServer2

If you configure HiveServer2 to use Kerberos authentication, HiveServer2 acquires a Kerberos ticket during startup. HiveServer2 requires a principal and keytab file specified in the configuration. Client applications (for example, JDBC or Beeline) must have a valid Kerberos ticket before initiating a connection to HiveServer2.

Configuring HiveServer2 for Kerberos-Secured Clusters

To enable Kerberos Authentication for HiveServer2, add the following properties in the `/etc/hive/conf/hive-site.xml` file:

```
<property>
  <name>hive.server2.authentication</name>
  <value>KERBEROS</value>
</property>
<property>
  <name>hive.server2.authentication.kerberos.principal</name>
  <value>hive/_HOST@YOUR-REALM.COM</value>
</property>
<property>
  <name>hive.server2.authentication.kerberos.keytab</name>
  <value>/etc/hive/conf/hive.keytab</value>
</property>
```

where:

- `hive.server2.authentication` is a client-facing property that controls the type of authentication HiveServer2 uses for connections to clients. In this case, HiveServer2 uses Kerberos to authenticate incoming clients.
- The `_HOST@YOUR-REALM.COM` value in the example above is the Kerberos principal for the host where HiveServer2 is running. The string _HOST in the properties is replaced at run time by the fully qualified domain name (FQDN) of the host machine where the daemon is running. Reverse DNS must be working on all the hosts configured this way. Replace `YOUR-REALM.COM` with the name of the Kerberos realm your Hadoop cluster is in.
- The `/etc/hive/conf/hive.keytab` value in the example above is a keytab file for that principal.

If you configure HiveServer2 to use both Kerberos authentication and secure impersonation, JDBC clients and Beeline can specify an alternate session user. If these clients have proxy user privileges, HiveServer2 impersonates the alternate user instead of the one connecting. The alternate user can be specified by the JDBC connection string `proxyUser=`*`userName`*

Configuring JDBC Clients for Kerberos Authentication with HiveServer2 (Using the Apache Hive Driver in Beeline)

JDBC-based clients must include `principal=<hive.server2.authentication.principal>` in the JDBC connection string. For example:

```
String url =
"jdbc:hive2://node1:10000/default;principal=hive/HiveServer2Host@YOUR-REALM.COM"
Connection con = DriverManager.getConnection(url);
```

where `hive` is the principal configured in `hive-site.xml` and `HiveServer2Host` is the host where HiveServer2 is running.

See the database drivers section on the Cloudera downloads web page to download and install the driver.

Using Beeline to Connect to a Secure HiveServer2

Use the following command to start `beeline` and connect to a secure HiveServer2 process. In this example, the HiveServer2 process is running on `localhost` at port `10000`:

```
$ /usr/lib/hive/bin/beeline
beeline> !connect
jdbc:hive2://localhost:10000/default;principal=hive/HiveServer2Host@YOUR-REALM.COM
0: jdbc:hive2://localhost:10000/default>
```

For more information about the Beeline CLI, see Using the Beeline CLI.

For instructions on encrypting communication with the ODBC/JDBC drivers, see Configuring Encrypted Communication Between HiveServer2 and Client Drivers.

Using LDAP Username/Password Authentication with HiveServer2

As an alternative to Kerberos authentication, you can configure HiveServer2 to use user and password validation backed by LDAP. The client sends a username and password during connection initiation. HiveServer2 validates these credentials using an external LDAP service.

You can enable LDAP Authentication with HiveServer2 using Active Directory or OpenLDAP.

Important: When using LDAP username/password authentication with HiveServer2, you must enable encrypted communication between HiveServer2 and its client drivers to avoid sending cleartext passwords. For instructions, see Configuring Encrypted Communication Between HiveServer2 and Client Drivers. To avoid sending LDAP credentials over a network in cleartext, see Configuring LDAPS Authentication with HiveServer2 on page 138.

Enabling LDAP Authentication with HiveServer2 using Active Directory

- **For managed clusters, use Cloudera Manager:**
  1. In the Cloudera Manager Admin Console, click **Hive** in the list of components, and then select the **Configuration** tab.
  2. Type "ldap" in the Search text box to locate the LDAP configuration fields.
  3. Check **Enable LDAP Authentication**.
  4. Enter the **LDAP URL** in the format `ldap[s]://<host>:<port>`
  5. Enter the **Active Directory Domain** for your environment.
  6. Click **Save Changes**.
- **For unmanaged clusters, use the command line:**

  Add the following properties to the `hive-site.xml`:

```
<property>
  <name>hive.server2.authentication</name>
  <value>LDAP</value>
</property>
<property>
  <name>hive.server2.authentication.ldap.url</name>
  <value>LDAP_URLL</value>
</property>
<property>
  <name>hive.server2.authentication.ldap.Domain</name>
  <value>AD_DOMAIN_ADDRESS</value>
</property>
```

Where:

The *LDAP_URL* value is the access URL for your LDAP server. For example, `ldap[s]://<host>:<port>`

Enabling LDAP Authentication with HiveServer2 using OpenLDAP

To enable LDAP authentication using OpenLDAP, include the following properties in `hive-site.xml`:

```
<property>
  <name>hive.server2.authentication</name>
  <value>LDAP</value>
</property>
<property>
  <name>hive.server2.authentication.ldap.url</name>
  <value>LDAP_URL</value>
</property>
<property>
  <name>hive.server2.authentication.ldap.baseDN</name>
  <value>LDAP_BaseDN</value>
</property>
```

where:

- The `LDAP_URL` value is the access URL for your LDAP server.
- The `LDAP_BaseDN` value is the base LDAP DN for your LDAP server; for example, `ou=People,dc=example,dc=com`.

Configuring JDBC Clients for LDAP Authentication with HiveServer2

The JDBC client requires a connection URL as shown below.

JDBC-based clients must include `user=LDAP_Userid;password=LDAP_Password` in the JDBC connection string. For example:

```
String url
="jdbc:hive2://#<host>:#<port>/#<dbName>;ssl=true;sslTrustStore=#<ssl_truststore_path>;trustStorePassword=#<truststore_password>"
Connection con = DriverManager.getConnection(url);
```

where the `LDAP_Userid` value is the user ID and `LDAP_Password` is the password of the client user.

Enabling LDAP Authentication for HiveServer2 in Hue

Enable LDAP authentication with HiveServer2 by setting the following properties under the `[beeswax]` section in `hue.ini`.

`auth_username`	LDAP username of Hue user to be authenticated.
`auth_password`	LDAP password of Hue user to be authenticated.

Hive uses these login details to authenticate to LDAP. The Hive service trusts that Hue has validated the user being impersonated.

Configuring LDAPS Authentication with HiveServer2

HiveServer2 supports LDAP username/password authentication for clients. Clients send LDAP credentials to HiveServer2 which in turn verifies them against the configured LDAP provider, such as OpenLDAP or Microsoft Active Directory. Most implementations now support LDAPS (LDAP over TLS/SSL), an authentication protocol that uses TLS/SSL to encrypt communication between the LDAP service and its client (in this case, HiveServer2) to avoid sending LDAP credentials in cleartext.

To configure the LDAPS service with HiveServer2:

1. Import the LDAP server CA certificate or the server certificate into a truststore on the HiveServer2 host. If you import the CA certificate, HiveServer2 will trust any server with a certificate issued by the LDAP server's CA. If you only import the server certificate, HiveServer2 trusts only that server. See Understanding Keystores and Truststores for more details.
2. Make sure the truststore file is readable by the `hive` user.
3. Set the `hive.server2.authentication.ldap.url` configuration property in `hive-site.xml` to the LDAPS URL. For example, `ldaps://sample.myhost.com`.

   Note: The URL scheme should be `ldaps` and *not* `ldap`.

4. If this is a managed cluster, in Cloudera Manager, go to the Hive service and select **Configuration**. Under Category, select **Security**. In the right panel, search for **HiveServer2 TLS/SSL Certificate Trust Store File**, and add the path to the truststore file that you created in step 1.

   If you are using an unmanaged cluster, set the environment variable HADOOP_OPTS as follows:

```
HADOOP_OPTS="-Djavax.net.ssl.trustStore=<trustStore-file-path>
-Djavax.net.ssl.trustStorePassword=<trustStore-password>"
```

5. Restart HiveServer2.

Pluggable Authentication

Pluggable authentication allows you to provide a custom authentication provider for HiveServer2.

To enable pluggable authentication:

1. Set the following properties in `/etc/hive/conf/hive-site.xml`:

```
<property>
  <name>hive.server2.authentication</name>
  <value>CUSTOM</value>
  <description>Client authentication types.
  NONE: no authentication check
  LDAP: LDAP/AD based authentication
  KERBEROS: Kerberos/GSSAPI authentication
  CUSTOM: Custom authentication provider
  (Use with property hive.server2.custom.authentication.class)
  </description>
</property>

<property>
  <name>hive.server2.custom.authentication.class</name>
  <value>pluggable-auth-class-name</value>
  <description>
  Custom authentication class. Used when property
  'hive.server2.authentication' is set to 'CUSTOM'. Provided class
  must be a proper implementation of the interface
  org.apache.hive.service.auth.PasswdAuthenticationProvider. HiveServer2
  will call its Authenticate(user, passed) method to authenticate requests.
  The implementation may optionally extend the Hadoop's
  org.apache.hadoop.conf.Configured class to grab Hive's Configuration object.
  </description>
</property>
```

2. Make the class available in the CLASSPATH of HiveServer2.

Trusted Delegation with HiveServer2

HiveServer2 determines the identity of the connecting user from the authentication subsystem (Kerberos or LDAP). Any new session started for this connection runs on behalf of this connecting user. If the server is configured to proxy the user at the Hadoop level, then all MapReduce jobs and HDFS accesses will be performed with the identity of the

connecting user. If Apache Sentry is configured, then this connecting userid can also be used to verify access rights to underlying tables and views.

Users with Hadoop superuser privileges can request an alternate user for the given session. HiveServer2 checks that the connecting user can proxy the requested userid, and if so, runs the new session as the alternate user. For example, the Hadoop superuser hue can request that a connection's session be run as user bob.

Alternate users for new JDBC client connections are specified by adding the `hive.server2.proxy.user=`*`alternate_user_id`* property to the JDBC connection URL. For example, a JDBC connection string that lets user hue run a session as user bob would be as follows:

```
# Login as super user Hue
kinit hue -k -t hue.keytab hue@MY-REALM.COM

# Connect using following JDBC connection string
#
jdbc:hive2://myHost.myOrg.com:10000/default;principal=hive/_HOST@MY-REALM.COM;hive.server2.proxy.user=bob
```

The connecting user must have Hadoop-level proxy privileges over the alternate user.

HiveServer2 Impersonation

> **Important:** This is not the recommended method to implement HiveServer2 authorization. Cloudera recommends you use Sentry to implement this instead.

HiveServer2 impersonation lets users execute queries and access HDFS files as the connected user rather than as the super user. Access policies are applied at the file level using the HDFS permissions specified in ACLs (access control lists). Enabling HiveServer2 impersonation bypasses Sentry from the end-to-end authorization process. Specifically, although Sentry enforces access control policies on tables and views within the Hive warehouse, it does not control access to the HDFS files that underlie the tables. This means that users without Sentry permissions to tables in the warehouse may nonetheless be able to bypass Sentry authorization checks and execute jobs and queries against tables in the warehouse as long as they have permissions on the HDFS files supporting the table.

To configure Sentry correctly, restrict ownership of the Hive warehouse to `hive:hive` and disable Hive impersonation.

To enable impersonation in HiveServer2:

1. Add the following property to the `/etc/hive/conf/hive-site.xml` file and set the value to `true`. (The default value is `false`.)

```
<property>
  <name>hive.server2.enable.impersonation</name>
  <description>Enable user impersonation for HiveServer2</description>
  <value>true</value>
</property>
```

2. In HDFS or MapReduce configurations, add the following property to the `core-site.xml` file:

```
<property>
  <name>hadoop.proxyuser.hive.hosts</name>
  <value>*</value>
</property>
<property>
  <name>hadoop.proxyuser.hive.groups</name>
  <value>*</value>
</property>
```

See also File System Permissions.

Securing the Hive Metastore

> **Note:** This is not the recommended method to protect the Hive Metastore. Cloudera recommends you use Sentry to implement this instead.

To prevent users from accessing the Hive metastore and the Hive metastore database using any method other than through HiveServer2, the following actions are recommended:

- Add a firewall rule on the metastore service host to allow access to the metastore port only from the HiveServer2 host. You can do this using iptables.
- Grant access to the metastore database only from the metastore service host. This is specified for MySQL as:

```
GRANT ALL PRIVILEGES ON metastore.* TO 'hive'@'metastorehost';
```

  where `metastorehost` is the host where the metastore service is running.

- Make sure users who are not admins cannot log on to the host on which HiveServer2 runs.

Disabling the Hive Security Configuration

Hive's security related metadata is stored in the configuration file `hive-site.xml`. The following sections describe how to disable security for the Hive service.

Disable Client/Server Authentication

To disable client/server authentication, set `hive.server2.authentication` to `NONE`. For example,

```
<property>
  <name>hive.server2.authentication</name>
  <value>NONE</value>
  <description>
    Client authentication types.
       NONE: no authentication check
       LDAP: LDAP/AD based authentication
       KERBEROS: Kerberos/GSSAPI authentication
       CUSTOM: Custom authentication provider
               (Use with property hive.server2.custom.authentication.class)
  </description>
</property>
```

Disable Hive Metastore security

To disable Hive Metastore security, perform the following steps:

- Set the `hive.metastore.sasl.enabled` property to `false` in all configurations, the metastore service side as well as for all clients of the metastore. For example, these might include HiveServer2, Impala, Pig and so on.
- Remove or comment the following parameters in `hive-site.xml` for the metastore service. Note that this is a server-only change.
  - `hive.metastore.kerberos.keytab.file`
  - `hive.metastore.kerberos.principal`

Disable Underlying Hadoop Security

If you also want to disable the underlying Hadoop security, remove or comment out the following parameters in `hive-site.xml`.

- `hive.server2.authentication.kerberos.keytab`
- `hive.server2.authentication.kerberos.principal`

Hive Metastore Server Security Configuration

Important:

This section describes how to configure security for the Hive metastore server. If you are using HiveServer2, see HiveServer2 Security Configuration.

Here is a summary of Hive metastore server security in CDH 5:

- No additional configuration is required to run Hive on top of a security-enabled Hadoop cluster in standalone mode using a local or embedded metastore.
- HiveServer does not support Kerberos authentication for clients. While it is possible to run HiveServer with a secured Hadoop cluster, doing so creates a security hole since HiveServer does not authenticate the Thrift clients that connect to it. Instead, you can use HiveServer2 HiveServer2 Security Configuration.
- The Hive metastore server supports Kerberos authentication for Thrift clients. For example, you can configure a standalone Hive metastore server instance to force clients to authenticate with Kerberos by setting the following properties in the `hive-site.xml` configuration file used by the metastore server:

```
<property>
  <name>hive.metastore.sasl.enabled</name>
  <value>true</value>
  <description>If true, the metastore thrift interface will be secured with SASL. Clients
 must authenticate with Kerberos.</description>
</property>

<property>
  <name>hive.metastore.kerberos.keytab.file</name>
  <value>/etc/hive/conf/hive.keytab</value>
  <description>The path to the Kerberos Keytab file containing the metastore thrift
server's service principal.</description>
</property>

<property>
  <name>hive.metastore.kerberos.principal</name>
  <value>hive/_HOST@YOUR-REALM.COM</value>
  <description>The service principal for the metastore thrift server. The special string
 _HOST will be replaced automatically with the correct host name.</description>
</property>
```

Note:

The values shown above for the `hive.metastore.kerberos.keytab.file` and `hive.metastore.kerberos.principal` properties are examples which you will need to replace with the appropriate values for your cluster. Also note that the Hive keytab file should have its access permissions set to `600` and be owned by the same account that is used to run the Metastore server, which is the `hive` user by default.

- Requests to access the metadata are fulfilled by the Hive metastore impersonating the requesting user. This includes read access to the list of databases, tables, properties of each table such as their HDFS location and file type. You can restrict access to the Hive metastore service by allowing it to impersonate only a subset of Kerberos users. This can be done by setting the `hadoop.proxyuser.hive.groups` property in `core-site.xml` on the Hive metastore host.

  For example, if you want to give the `hive` user permission to impersonate members of groups `hive` and `user1`:

```
<property>
<name>hadoop.proxyuser.hive.groups</name>
<value>hive,user1</value>
</property>
```

In this example, the Hive metastore can impersonate users belonging to *only* the `hive` and `user1` groups. Connection requests from users not belonging to these groups will be rejected.

Using Hive to Run Queries on a Secure HBase Server

To use Hive to run queries on a secure HBase Server, you must set the following `HIVE_OPTS` environment variable:

```
env HIVE_OPTS="-hiveconf hbase.security.authentication=kerberos -hiveconf
hbase.master.kerberos.principal=hbase/_HOST@YOUR-REALM.COM -hiveconf
hbase.regionserver.kerberos.principal=hbase/_HOST@YOUR-REALM.COM -hiveconf
hbase.zookeeper.quorum=zookeeper1,zookeeper2,zookeeper3" hive
```

where:

- You replace `YOUR-REALM` with the name of your Kerberos realm
- You replace `zookeeper1`, `zookeeper2`, `zookeeper3` with the names of your ZooKeeper servers. The `hbase.zookeeper.quorum` property is configured in the `hbase-site.xml` file.
- The special string `_HOST` is replaced at run-time by the fully qualified domain name of the host machine where the HBase Master or RegionServer is running. This requires that reverse DNS is properly working on all the hosts configured this way.

In the following, `_HOST` is the name of the host where the HBase Master is running:

```
-hiveconf hbase.master.kerberos.principal=hbase/_HOST@YOUR-REALM.COM
```

In the following, `_HOST` is the hostname of the HBase RegionServer that the application is connecting to:

```
-hiveconf hbase.regionserver.kerberos.principal=hbase/_HOST@YOUR-REALM.COM
```

Note:

You can also set the `HIVE_OPTS` environment variable in your shell profile.

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.
```