Hortonworks Data Platform

Data Access

(December 15, 2017)

docs.cloudera.com

Hortonworks Data Platform: Data Access

Copyright © 2012-2017 Hortonworks, Inc. Some rights reserved.

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

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

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

Except where otherwise noted, this document is licensed under
Creative Commons Attribution ShareAlike 4.0 License.
http://creativecommons.org/licenses/by-sa/4.0/legalcode

Table of Contents

List of Figures

List of Tables

List of Examples

1. What's New in Data Access for HDP 2.6

New features and changes for Data Access components have been introduced in Hortonworks Data Platform (HDP), version 2.6, along with content updates. The new and changed features and documentation updates are described in the following sections.

- What's New in Apache Hive [1]
- What's New in Apache Tez [2]
- What's New in Apache HBase [2]
- What's New in Apache Phoenix [2]
- Druid [3]

Important

HDP features that are labeled *Technical Preview* are considered under development. Do not use Technical Preview features in your production systems. If you have questions regarding such a feature, contact Support by logging a case on the Hortonworks Support Portal.

1.1. What's New in Apache Hive

Important new features of Apache Hive in Hortonworks Data Platform (HDP) 2.6 include the following:

Full Support and Easier Setup for Interactive SQL Queries

Hortonworks supports interactive SQL queries with Hive low-latency analytical processing (LLAP). In the previous HDP release, Hive LLAP was a Technical Preview. Persistent query infrastructure and optimized data caching enable Hive LLAP. See Setting Up Hive LLAP for how to enable Hive LLAP in Apache Ambari.

LLAP-Focused Revision of the HDP Apache Hive Performance Tuning Guide

The *Hive Performance Tuning Guide* focuses on Hive LLAP to leverage gains in performant data analytics technologies. The guide includes a diverse set of recommendations that are tailored to optimizing the Tez framework for efficient querying of a Hive EDW. While interactive querying is emphasized, the documentation includes guidelines for environments that prefer or must use Hive on Tez without LLAP.

Support for the ACID-Transaction MERGE Statement

Hive in HDP 2.6 supports the MERGE SQL statement, which simplifies updating data, deleting data, and change data captures among tables.

Enhancements for SQL and TPC-DS Completeness

You can offload SQL workload to Hive using all TPC-DS queries with only trivial rewrites. The addition of multiple and scalar subqueries, INTERSECT and EXCEPT operators, standard

syntax for ROLLUP / GROUPING SET, and syntax improvements for GROUP BY and ORDER BY make the SQL offloading easier than before.

Hive View 2.0

With the release of Apache Ambari 2.5.0, HDP provides Hive View 2.0. Hive View 2.0 is an enhancement over earlier Hive View UIs because it has more nimble ways to create and manage databases and tables, offers visual explain plans of your queries that pinpoint workload costs, and provides a way to view security policies at the table level.

1.2. What's New in Apache Tez

Expansion of Tez View Capabilities

Tez View in Apache Ambari 2.5.0 has more features to debug Hive queries and Apache Pig scripts. Among the main improvements are more powerful query search capabilities and a new Total Timeline view that provides visualizations and specific data about where query time is spent.

1.3. What's New in Apache HBase

HBase in Hortonworks Data Platform (HDP) 2.6 includes the following new features:

HBase Storage Quota (Technical Preview)

In a multitenant environment, you often want to set quotas for limited resources for networking and storage to protect the SLAs of critical workloads. Earlier versions of HBase that were bundled in HDP support setting quota limits on RPC requests, also known as request throttling. HBase in HDP 2.6 introduces storage quota. This allows you to manage storage at either the namespace or the table level.

HBase Backup-and-Restore Supports Bulk-Loaded Data (Technical Preview)

HDP 2.6 allows you to use incremental backups with bulk-loaded data. In HDP 2.5, bulk-loaded data is only included in full backups. Bulk loading is a common technique for ingesting data into HBase. Bulk-loaded data does not produce write-ahead-logs (WALs).

1.4. What's New in Apache Phoenix

Phoenix in Hortonworks Data Platform (HDP) 2.6 includes the following new features:

Phoenix Data Integrity Tool (Technical Preview)

Phoenix in HDP 2.6 has a data integrity tool to help you troubleshoot serious problems with the system and to assist in putting Phoenix in good working condition. The tool runs a group of diagnostics, ranging from a check to verify the existence of the SYSTEM.CATALOG table to testing the index table's availability.

Phoenix Indexing Improvements

Phoenix in HDP 2.6 has multiple improvements in indexing stability and performance. The indexing tool supports incremental rebuilds and the overall indexing speed is lot faster. The

enhancements support part of Phoenix's mission to provide better secondary indexing for HBase data.

1.5. Druid

Druid in the HDP Stack

Druid is a high-performance, distributed datastore that delivers subsecond OLAP queries even when you have terabytes of data and dozens of dimensions. In Hortonworks Data Platform (HDP), Druid can be installed and configured with Ambari, just like any other HDP component. Druid can be integrated with Apache Hive, allowing subsecond SQL queries using any Hive-compatible tool.

2. Data Warehousing with Apache Hive

Hortonworks Data Platform deploys Apache Hive for your Hadoop cluster.

Hive is a data warehouse infrastructure built on top of Hadoop. It provides tools to enable easy data ETL, a mechanism to put structures on the data, and the capability for querying and analysis of large data sets stored in Hadoop files.

Hive defines a simple SQL query language, called *HiveQL*, that enables users familiar with SQL to query the data. At the same time, this language also allows programmers who are familiar with the MapReduce framework to be able to plug in their custom mappers and reducers to perform more sophisticated analysis that may not be supported by the built-in capabilities of the language.

In this document:

2.1. Content Roadmap

This roadmap provides links to the available content resources for Apache Hive.

Table 2.1. Hive Content roadmap

Task	Resources	Source	Description
Understanding	Presentations and Papers about Hive	Apache wiki	Contains meeting notes, presentations, and whitepapers from the Apache community.
Getting Started	Hive Tutorial	Apache wiki	Provides a basic overview of Apache Hive and contains some examples on working with tables, loading data, and querying and inserting data.
	Hive Tutorial	Hortonworks	Uses the Ambari HDFS file view to store massive data files of statistics. Implements implement Hive queries to analyze, process, and filter that data.
Installing and Upgrading	Ambari Automated Install Guide	Hortonworks	Ambari provides an end-to-end management and monitoring solution for your HDP cluster. Using the Ambari Web UI and REST APIs, you can deploy, operate, manage configuration changes, and monitor services for all nodes in your cluster from a central point.
	Non-Ambari Cluster Installation Guide	Hortonworks	Describes the information and materials you need to get ready to install the Hortonworks Data Platform (HDP) manually.

Task	Resources	Source	Description
	Ambari Upgrade Guide	Hortonworks	Ambari and the HDP Stack being managed by Ambari can be upgraded independently. This guide provides information on: Getting ready to upgrade Ambari and HDP, Upgrading Ambari, and Upgrading HDP.
	Non-Ambari Cluster Upgrade Guide	Hortonworks	These instructions cover the upgrade between two minor releases. If you need to upgrade between two maintenance releases, follow the upgrade instructions in the HDP Release Notes.
	Installing Hive	Apache wiki	Describes how to install Apache Hive separate from the HDP environment.
	Configuring Hive	Apache wiki	Describes how to configure Apache Hive separate from the HDP environment. Also useful for troubleshooting Hive in HDP.
Administering	Creating your Hive View instance	Hortonworks	Describes how to set up and create the Hive 1.0 or Hive 1.5 instance in Ambari.
	Setting Up the Metastore	Apache wiki	Describes the metastore parameters.
	Setting Up Hive Web Interface	Apache wiki	Describes the Hive Web Interface, an alternative to using the Hive CLI, its features, configuration and some tips and tricks for using.
	Setting Up Hive Server	Apache wiki	Describes how to set up the server. How to use a client with this server is described in the HiveServer2 Clients document.
Developing	Using Hive View 2.0	Hortonworks	Shows how to use the Hive view to browse databases, write and execute queries, and manage jobs and history.
	Moving data into Hive	Hortonworks	Shows the multiple ways to move data into Hive.
	Hive Operators and Functions	Apache wiki	Describes the Language Manual UDF.
	Beeline: HiveServer2 Client	Apache wiki	Describes how to use the Beeline client.
Security	Hadoop Security Guide	Hortonworks	Provides details of the security features implemented in the Hortonworks Data Platform (HDP).
Scheduling Workflow	Using HDP for Workflow and Scheduling with Oozie	Hortonworks	Oozie is a server-based workflow engine specialized in running workflow jobs with actions that execute Hadoop jobs, such as MapReduce, Pig, Hive, Sqoop, HDFS operations, and sub-workflows.
High Availability	High Availability for Hive Metastore	Hortonworks	Provides details for system administrators who need to configure the Hive Metastore service for High Availability.
Performance Tuning	Hive Performance Tuning Guide	Hortonworks	Lists best practices for an HDP Hive cluster, both for users who run interactive queries and for users who analyze a Hive EDW with batch processing.
Interactive Queries with Apache Hive LLAP	Setting up Hive LLAP Hive LLAP on Your Cluster YouTube video: Enable Hive LLAP on HDP 2.6 for Interactive SQL	Hortonworks	Apache Hive enables interactive and sub-second SQL through low-latency analytical processing (LLAP)., a new component introduced in Hive 2.0 that makes Hive faster by using persistent query infrastructure and optimized data caching.
Hive-HBase Integration	HBaseIntegration wiki	Apache wiki	Describes how to integrate the two data access components so that HiveQL

Task	Resources	Source	Description
			statements can access HBase tables for both read (SELECT) and write (INSERT) operations.
Reference	Javadocs	Apache wiki	Language reference documentation available in the Apache wiki.
	SQL Language Manual	Apache wiki	
Contributing	Hive Developer FAQ	Apache wiki	Resources available if you want to contribute to the Apache community.
	How to Contribute	Apache wiki	
	Hive Developer Guide	Apache wiki	
	Plug-in Developer Kit	Apache wiki	
	Unit Test Parallel Execution	Apache wiki	
	Hive Architecture Overview	Apache wiki	
	Hive Design Docs	Apache wiki	
	Full-Text Search over All Hive Resources	Apache wiki	
	Project Bylaws	Apache wiki	
Other resources	Hive Mailing Lists	Apache wiki	Additional resources available.
	Hive on Amazon Web Services	Apache wiki	
	Hive on Amazon Elastic MapReduce	Apache wiki	

2.2. Features Overview

The following subsections provide brief descriptions of enhancements to Apache Hive versions 0.13 and 0.14 that are supported by the HDP distribution of Hive.

2.2.1. Temporary Tables

Temporary tables are supported in Hive 0.14 and later. A temporary table is a convenient way for an application to automatically manage intermediate data generated during a complex query. Rather than manually deleting tables needed only as temporary data in a complex query, Hive automatically deletes all temporary tables at the end of the Hive session in which they are created. The data in these tables is stored in the user's scratch directory rather than in the Hive warehouse directory. The scratch directory effectively acts as the data sandbox for a user, located by default in `/tmp/hive-<username>`.

Tip

See Apache Hive AdminManual Configuration for information about configuration to use a non-default scratch directory.

Hive users create temporary tables using the TEMPORARY keyword:

```
CREATE TEMPORARY TABLE tmp1 (c1 string);
CREATE TEMPORARY TABLE tmp2 AS ...
CREATE TEMPORARY TABLE tmp3 LIKE ...
```

Multiple Hive users can create multiple Hive temporary tables with the same name because each table resides in a separate session.

Temporary tables support most table options, but not all. The following features are not supported:

- Partitioned columns
- Indexes

Note

A temporary table with the same name as a permanent table causes all references to that table name to resolve to the temporary table. The user cannot access the permanent table during that session without dropping or renaming the temporary table.

2.2.2. Optimized Row Columnar (ORC) Format

ORC-based tables are supported in Hive 0.14.0 and later. These tables can contain more than 1,000 columns.

For more information about how the ORC file format enhances Hive performance, see the Maximizing Storage Resources chapter in the *HDP Apache Hive Performance Tuning Guide* and LanguageManual ORC on the Apache site.

2.2.3. SQL Optimization

Cost-based optimization (CBO) of SQL queries is supported in Hive 0.13.0 and later. CBO uses Hive table, table partition, and column statistics to create efficient query execution plans. Efficient query plans better utilize cluster resources and improve query latency. CBO is most useful for complex queries that contain multiple JOIN statements and for queries on very large tables.

Note

Tables are not required to have partitions to generate CBO statistics. Column-level CBO statistics can be generated by both partitioned and unpartitioned tables.

CBO generates the following statistics:

Statistics Granularity	Description
Table-level	- Uncompressed size of table - Number of rows - Number of files
Column-level	- Number of distinct values - Number of NULL values - Minimum value - Maximum value

CBO requires column-level statistics to generate the best query execution plans. Later, when viewing these statistics from the command line, you can choose to also include table-level

statistics that are generated by the `hive.stats.autogather` configuration property. However, CBO does not use these table-level statistics to generate query execution plans.

2.2.3.1. Enabling Cost-Based Optimization

About this Task

Turning on CBO is highly recommended.

Prerequisite

You must have administrator previleges.

Steps

1. In Ambari, open **Services** > **Hive** > **Configs** tab.
2. Refer to the following table for the properties that enable CBO and assist with generating table statistics, along with the required property settings.

   You can view the properties by either of these methods:

   Type each property name in the Filter field in the top right corner.
   Open the **General**, **Advanced hive-env**, etc., sections and scan the lists of each category.
3. Click **Save**.
4. If prompted to restart, restart the Hive Service.

Table 2.2. CBO Configuration Parameters

Configuration Parameter	Setting to Enable CBO	Description
`hive.cbo.enable`	**`true`**	Enables cost-based query optimization.
`hive.stats.autogather`	**`true`**	Enables automated gathering of table-level statistics for newly created tables and table partitions, such as tables created with the INSERT OVERWRITE statement. The parameter does not produce column-level statistics, such as those generated by CBO. If disabled, administrators must manually generate the table-level statistics for newly generated tables and table partitions with the ANALYZE TABLE statement.

2.2.3.2. Statistics

Tip

Gather both column and table statistics for best query performance.

Column and table statistics must be calculated for optimal Hive performance because they are critical for estimating predicate selectivity and cost of the plan. In the absence of table statistics, Hive CBO does not function. Certain advanced rewrites require column statistics.

Ensure that the configuration properties in the following table are set to `true` to improve the performance of queries that generate statistics. You can set the properties using Ambari or by customizing the `hive-site.xml` file.

Configuration Parameter	Setting to Enable Statistics	Description
`hive.stats.fetch.column.stats`	**`true`**	Instructs Hive to collect column-level statistics.
`hive.compute.query.using.stats`	**`true`**	Instructs Hive to use statistics when generating query plans.

2.2.3.2.1. Generating Hive Statistics

The ANALYZE TABLE command generates statistics for tables and columns. The following lines show how to generate different types of statistics on Hive objects.

Gathering table statistics for non-partitioned tables

```
ANALYZE TABLE [table_name] COMPUTE STATISTICS;
```

Gathering table statistics for partitioned tables

```
ANALYZE TABLE [table_name] PARTITION(partition_column) COMPUTE STATISTICS;
```

Gathering column statistics for the entire table

```
ANALYZE TABLE [table_name] COMPUTE STATISTICS for COLUMNS
 [comma_separated_column_list];
```

Gathering statistics for the **`partition2`** column on a table partitioned on **`col1`** with key **`x`**

```
ANALYZE TABLE partition2 (col1="x") COMPUTE STATISTICS for COLUMNS;
```

2.2.3.2.2. Viewing Generated Statistics

Use the DESCRIBE statement to view statistics generated by CBO. Include the EXTENDED keyword if you want to include statistics gathered when the `hive.stats.fetch.column.stats` and `hive.compute.query.using.stats` properties are enabled.

- Viewing Table Statistics
  - Use the following syntax to view table statistics:

    ```
    DESCRIBE [EXTENDED] table_name;
    ```

    Note

    The EXTENDED keyword can be used only if the `hive.stats.autogather` property is enabled in the `hive-site.xml` configuration file.

  - The following example displays all statistics for the employees table:

    ```
    DESCRIBE EXTENDED employees;
    ```

- If the table statistics are up-to-date, the output includes the following table parameter information:

```
{\"BASIC_STATS\":\"true\",\" ...
```

- Viewing Column Statistics
  - Use the following syntax to view column statistics:

```
DESCRIBE FORMATTED [db_name.]table_name.column_name;
```

  - The following example displays statistics for the region column in the employees table:

```
DESCRIBE FORMATTED employees.region;
```

  - If the table statistics are up-to-date, the output includes the following table parameter information:

```
COLUMN_STATS_ACCURATE
```

Note

See Statistics in Hive on the Apache website for more information.

2.2.4. SQL Compliance and ACID-Based Transactions

This section discusses the ongoing implementation of standard SQL syntax in Hive. Although SQL in Hive does not yet entirely support the SQL-2011 standard, versions 0.13 and 0.14 provide significant improvements to the parity between SQL as used in Hive and SQL as used in traditional relational databases.

2.2.4.1. Transactions in Hive

Support for transactions in Hive 0.13 and later enables SQL atomicity of operations at the row level rather than at the level of a table or partition. This allows a Hive client to read from a partition at the same time that another Hive client is adding rows to the same partition. In addition, transactions provide a mechanism for streaming clients to rapidly update Hive tables and partitions. Hive transactions differ from RDBMS transactions in that each transaction has an identifier, and multiple transactions are grouped into a single transaction batch. A streaming client requests a set of transaction IDs after connecting to Hive and subsequently uses these transaction IDs one at a time during the initialization of new transaction batches. Clients write one or more records for each transaction and either commit or abort a transaction before moving to the next transaction.

ACID is an acronym for four required traits of database transactions: atomicity, consistency, isolation, and durability.

Transaction Attribute	Description
Atomicity	An operation either succeeds completely or fails; it does not leave partial data.
Consistency	Once an application performs an operation, the results of that operation are visible to the application in every subsequent operation.

Transaction Attribute	Description
Isolation	Operations by one user do not cause unexpected side effects for other users.
Durability	Once an operation is complete, it is preserved in case of machine or system failure.

Administrators:

To use ACID-based transactions, administrators must use a transaction manager that supports ACID and the ORC file format. See Understanding and Administering Hive Compactions for instructions on configuring a transaction manager for Hive.

Developers:

Developers and others can create ACID tables by either of the following methods:

Creating Hive ACID Transaction Tables in Ambari
Creating the tables with SQL outside the Ambari framework

Note

See the Hive wiki for more information about Hive's support of ACID semantics for transactions.

2.2.4.1.1. Understanding and Administering Hive Compactions

Hive stores data in base files that cannot be updated by HDFS. Instead, Hive creates a set of delta files for each transaction that alters a table or partition and stores them in a separate delta directory. Occasionally, Hive *compacts*, or merges, the base and delta files. Hive performs all compactions in the background without affecting concurrent reads and writes of Hive clients. There are two types of compactions:

Table 2.3. Hive Compaction Types

Compaction Type	Description
Minor	Rewrites a set of delta files to a single delta file for a bucket.
Major	Rewrites one or more delta files and the base file as a new base file for a bucket.

By default, Hive automatically compacts delta and base files at regular intervals. However, Hadoop administrators can configure automatic compactions, as well as perform manual compactions of base and delta files using the following configuration parameters in `hive-site.xml`.

Table 2.4. Hive Transaction Configuration Parameters

Configuration Parameter	Description
`hive.txn.manager`	Specifies the class name of the transaction manager used by Hive. Set this property to `org.apache.hadoop.hive.ql.lockmgr.DbTxnManager` to enable transactions. The default value is `org.apache.hadoop.hive.ql.lockmgr.DummyTxnManager`, which disables transactions.

Configuration Parameter	Description
hive.compactor.initiator.on	Specifies whether to run the initiator and cleaner threads on this Metastore instance. The default value is **false**. Must be set to **true** for exactly one instance of the Hive metastore service.
hive.compactor.worker.threads	Specifies the number of of worker threads to run on this Metastore instance. The default value is 0, which must be set to greater than 0 to enable compactions. Worker threads initialize MapReduce jobs to do compactions. Increasing the number of worker threads decreases the time required to compact tables after they cross a threshold that triggers compactions. However, increasing the number of worker threads also increases the background load on a Hadoop cluster.
hive.compactor.worker.timeout	Specifies the time period, in seconds, after which a compaction job is failed and re-queued. The default value is 86400 seconds, or 24 hours.
hive.compactor.check.interval	Specifies the time period, in seconds, between checks to see if any partitions require compacting. The default value is 300 seconds. Decreasing this value reduces the time required to start a compaction for a table or partition. However, it also increases the background load on the NameNode since each check requires several calls to the NameNode.
hive.compactor.delta.num.threshold	Specifies the number of delta directories in a partition that triggers an automatic minor compaction. The default value is 10.
hive.compactor.delta.pct.threshold	Specifies the percentage size of delta files relative to the corresponding base files that triggers an automatic major compaction. The default value is.1, which is 10 percent.
hive.compactor.abortedtxn.threshold	Specifies the number of aborted transactions on a single partition that trigger an automatic major compaction.

2.2.4.1.2. Configuring the Hive Transaction Manager

Configure the following Hive properties to enable transactions:

- hive.txn.manager
- hive.compactor.initiator.on
- hive.compactor.worker.threads

Tip

To disable automatic compactions for individual tables, set the NO_AUTO_COMPACTION table property for those tables. This overrides the configuration settings in the hive-site.xml file. However, this property does not prevent manual compactions.

If you experience problems while enabling Hive transactions, check the Hive log file at /tmp/hive/hive.log.

2.2.4.1.3. Performing Manual Compactions

Hive administrators use the ALTER TABLE DDL command to queue requests that compact base and delta files for a table or partition:

```
ALTER TABLE tablename [PARTITION (partition_key='partition_value' [,...])]
 COMPACT 'compaction_type'
```

Use the SHOW COMPACTIONS command to monitor the progress of the compactions:

```
SHOW COMPACTIONS
```

Note

ALTER TABLE will compact tables even if the NO_AUTO_COMPACTION table property is set.

The SHOW COMPACTIONS command provides the following output for each compaction:

- Database name
- Table name
- Partition name
- Major or minor compaction
- Compaction state:
  - Initiated - waiting in queue
  - Working - currently compacting
  - Ready for cleaning - compaction completed and old files scheduled for removal
- Thread ID
- Start time of compaction

Hive administrators can also view a list of currently open and aborted transactions with the the SHOW TRANSACTIONS command. This command provides the following output for each transaction:

- Transaction ID
- Transaction state
- Hive user who initiated the transaction
- Host machine where transaction was initiated

2.2.4.1.4. Lock Manager

DbLockManager, introduced in Hive 0.13, stores all transaction and related lock information in the Hive Metastore. Heartbeats are sent regularly from lock holders and transaction initiators to the Hive metastore to prevent stale locks and transactions. The lock or transaction is aborted if the metastore does not receive a heartbeat within the amount of time specified by the hive.txn.timeout configuration property. Hive administrators use the SHOW LOCKS DDL command to view information about locks associated with transactions.

This command provides the following output for each lock:

- Database name
- Table name
- Partition, if the table is partitioned
- Lock state:
  - Acquired - transaction initiator hold the lock
  - Waiting - transaction initiator is waiting for the lock
  - Aborted - the lock has timed out but has not yet been cleaned
- Lock type:
  - Exclusive - the lock may not be shared
  - Shared_read - the lock may be shared with any number of other shared_read locks
  - Shared_write - the lock may be shared by any number of other shared_read locks but not with other shared_write locks
- Transaction ID associated with the lock, if one exists
- Last time lock holder sent a heartbeat
- Time the lock was acquired, if it has been acquired
- Hive user who requested the lock
- Host machine on which the Hive user is running a Hive client

Note

The output of the command reverts to behavior prior to Hive 0.13 if administrators use ZooKeeper or in-memory lock managers.

2.2.4.1.5. Transaction Limitations

HDP currently has the following limitations for ACID-based transactions in Hive:

- The BEGIN, COMMIT, and ROLLBACK SQL statements are not yet supported. All operations are automatically committed as transactions.
- The user initiating the Hive session must have WRITE permission for the destination partition or table.
- ZooKeeper and in-memory locks are not compatible with transactions.
- Only ORC files are supported.
- Destination tables must be bucketed and not sorted.

- The only supported isolation level is Snapshot.

2.2.4.2. INSERT ... VALUES, UPDATE, DELETE, and MERGE SQL Statements

INSERT ... VALUES, UPDATE, DELETE, and MERGE SQL statements are supported in Apache Hive 0.14 and later. The INSERT ... VALUES statement enables users to write data to Apache Hive from values provided in SQL statements. The UPDATE and DELETE statements enable users to modify and delete values already written to Hive. The MERGE statement streamlines UPDATEs, DELETEs, and change data capture operations by drawing on coexisting tables. All four statements support auto-commit, which means that each statement is a separate transaction that is automatically committed after the SQL statement is executed.

The INSERT ... VALUES, UPDATE, and DELETE statements require the following property values in the hive-site.xml configuration file:

Configuration Property	Required Value
`hive.enforce.bucketing`	`true`
`hive.exec.dynamic.partition.mode`	`nonstrict`

Note

Administrators must use a transaction manager that supports ACID and the ORC file format to use transactions. See Hive Transactions for information about configuring other properties related to use ACID-based transactions.

INSERT ... VALUES Statement

The INSERT ... VALUES statement is revised to support adding multiple values into table columns directly from SQL statements. A valid INSERT ... VALUES statement must provide values for each column in the table. However, users may assign null values to columns for which they do not want to assign a value. In addition, the PARTITION clause must be included in the DML.

```
INSERT INTO TABLE tablename [PARTITION (partcol1=val1, partcol2=val2 ...)]
 VALUES values_row [, values_row...]
```

In this syntax, `values_row` is `(value [, value])` and where `value` is either `NULL` or any SQL literal.

The following example SQL statements demonstrate several usage variations of this statement:

```
CREATE TABLE students (name VARCHAR(64), age INT, gpa DECIMAL(3,2)) CLUSTERED
 BY (age) INTO 2 BUCKETS STORED AS ORC;
```

```
INSERT INTO TABLE students VALUES ('fred flintstone', 35, 1.28), ('barney
 rubble', 32, 2.32);
```

```
CREATE TABLE pageviews (userid VARCHAR(64), link STRING, from STRING)
 PARTITIONED BY (datestamp STRING) CLUSTERED BY (userid) INTO 256 BUCKETS
 STORED AS ORC;
```

```
INSERT INTO TABLE pageviews PARTITION (datestamp = '2014-09-23') VALUES
 ('jsmith', 'mail.com', 'sports.com'), ('jdoe', 'mail.com', null);
```

```
INSERT INTO TABLE pageviews PARTITION (datestamp) VALUES ('tjohnson',
 'sports.com', 'finance.com', '2014-09-23'), ('tlee', 'finance.com', null,
 '2014-09-21');
```

UPDATE Statement

Use the UPDATE statement to modify data already written to Apache Hive. Depending on the condition specified in the optional WHERE clause, an UPDATE statement may affect every row in a table. You must have both the SELECT and UPDATE privileges to use this statement.

```
UPDATE tablename SET column = value [, column = value ...] [WHERE
 expression];
```

The UPDATE statement has the following limitations:

- The expression in the WHERE clause must be an expression supported by a Hive SELECT clause.
- Partition and bucket columns cannot be updated.
- Query vectorization is automatically disabled for UPDATE statements. However, updated tables can still be queried using vectorization.
- Subqueries are not allowed on the right side of the SET statement.

The following example demonstrates the correct usage of this statement:

```
UPDATE students SET name = null WHERE gpa <= 1.0;
```

DELETE Statement

Use the DELETE statement to delete data already written to Apache Hive.

```
DELETE FROM tablename [WHERE expression];
```

The DELETE statement has the following limitation: query vectorization is automatically disabled for the DELETE operation. However, tables with deleted data can still be queried using vectorization.

The following example demonstrates the correct usage of this statement:

```
DELETE FROM students WHERE gpa <= 1,0;
```

MERGE Statement

Use the MERGE statement to efficiently perform record-level INSERT, UPDATE, and DELETE operations within Hive tables. The MERGE statement can be a key tool of Hadoop data management.

The MERGE statement is based on ANSI-standard SQL.

The following SQL statement is an example of valid MERGE usage:

```
merge into customer
using ( select * from new_customer_stage) sub
on sub.id = customer.id
when matched then update set name = sub.name, state = sub.new_state
when not matched then insert values (sub.id, sub.name, sub.state);
```

Tip

For a more in-depth discussion of practical applications of the MERGE statement, see the blog Apache Hive: Moving Beyond Analytics Offload with SQL MERGE.

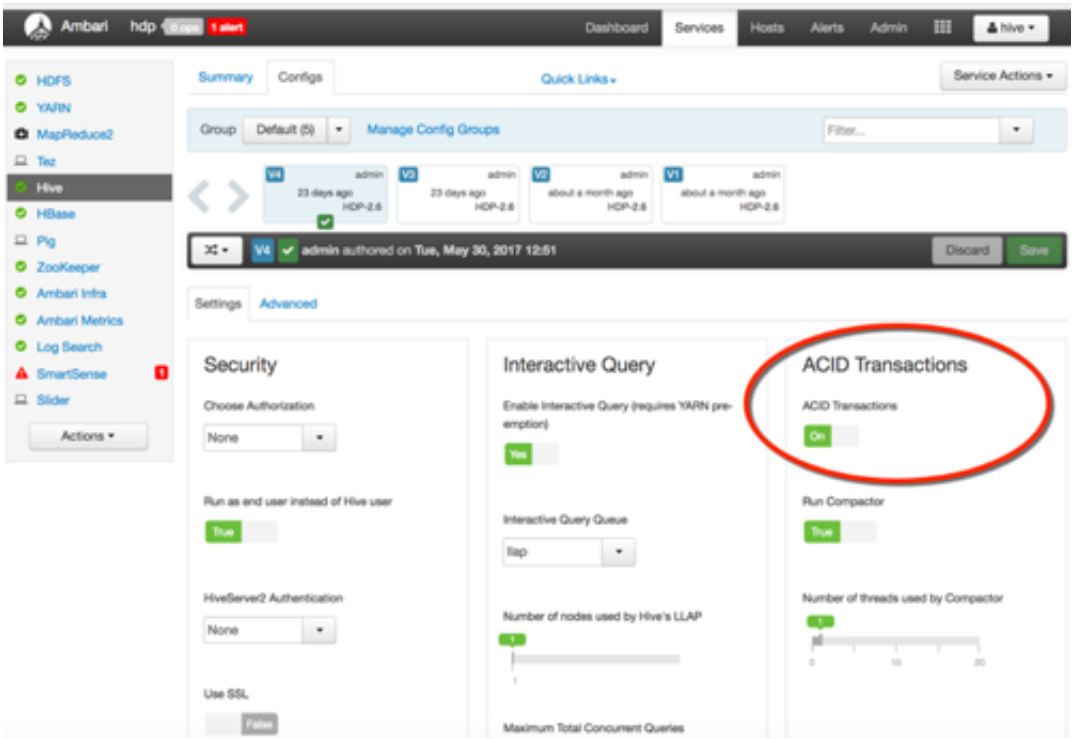
2.2.4.3. Creating Hive ACID Transaction Tables

About this Task

Using Hive View 2.0 of Ambari, you can enable ACID-based transactions so that you can create tables that support transactional SQL. Hive View 2.0 also lets you export the DDL of the table to fully functional scripts.

Steps

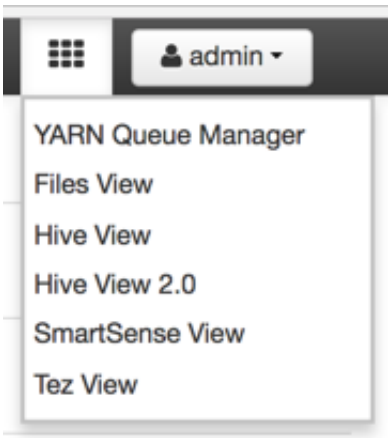
1. In Ambari, select **Services** > **Hive** > **Configs**.
2. On the **Settings** subtab, set the **ACID Transactions** slider to **On**:



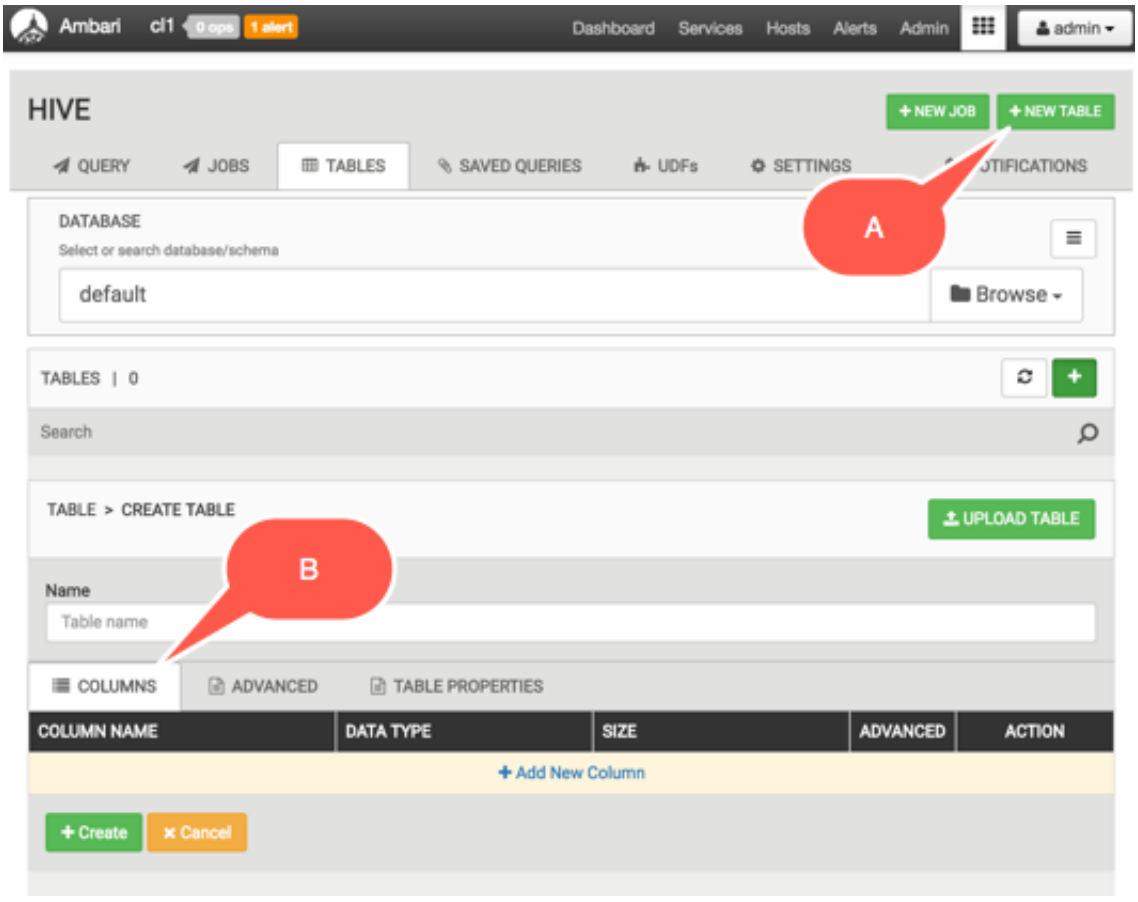
3. Click **Save**.
4. Enter an annotation about the change.

5. If the **Dependent Configurations** window appears, review the recommended changes. You can modify the settings if you need to do so for your cluster environment.

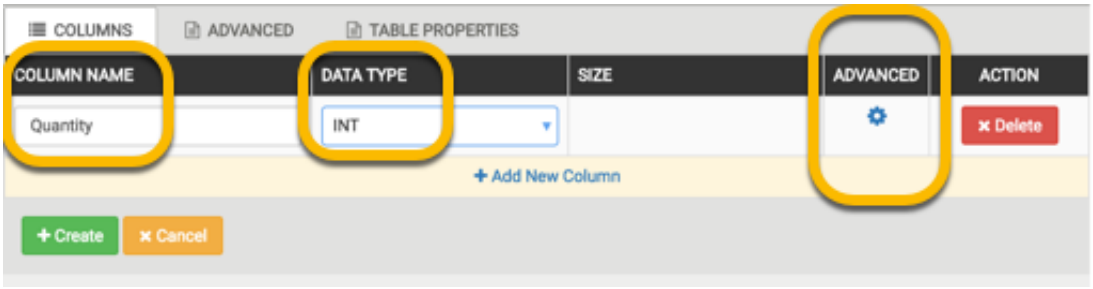
6. Open Hive View 2.0. One way that you can do this is by clicking the palette icon in the upper right corner of the Ambari window, as shown in the following screenshot.



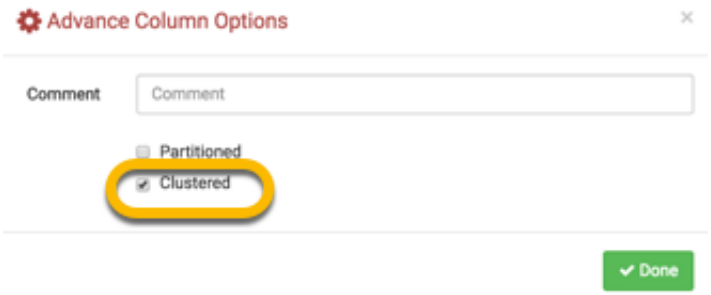
7. Select **+NEW TABLE** button > **COLUMNS** subtab.



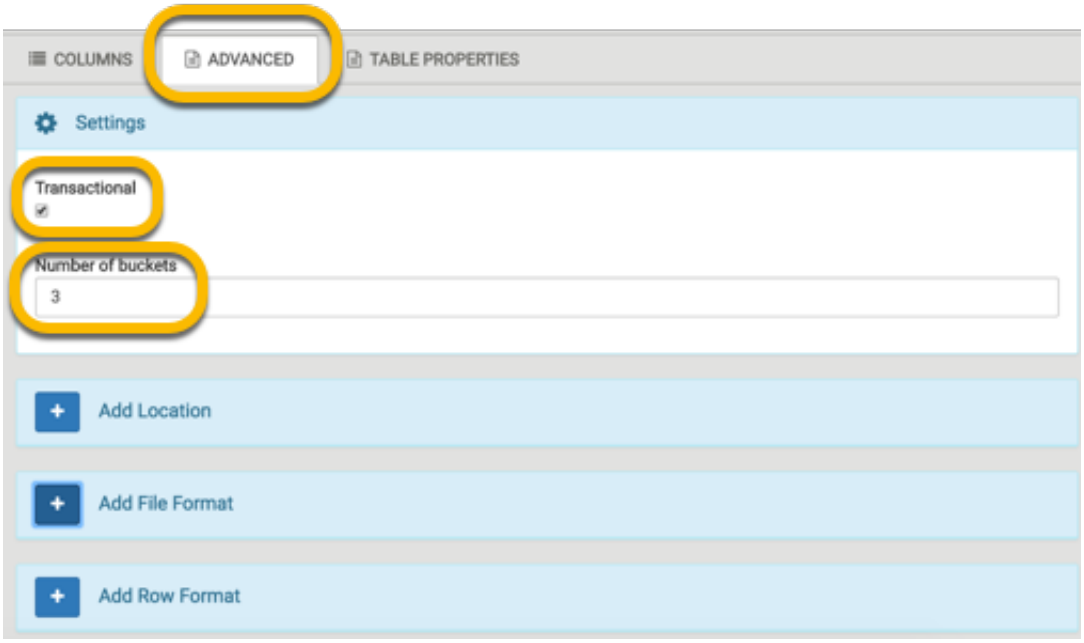
8. Define the column name and data type for each column of your table. Click the ADVANCED configuration wheel for the column to be the key column.



9. After clicking the ADVANCED configuration wheel for the column to be set as the key column, designate the clustering key by clicking the **Clustered** checkbox.



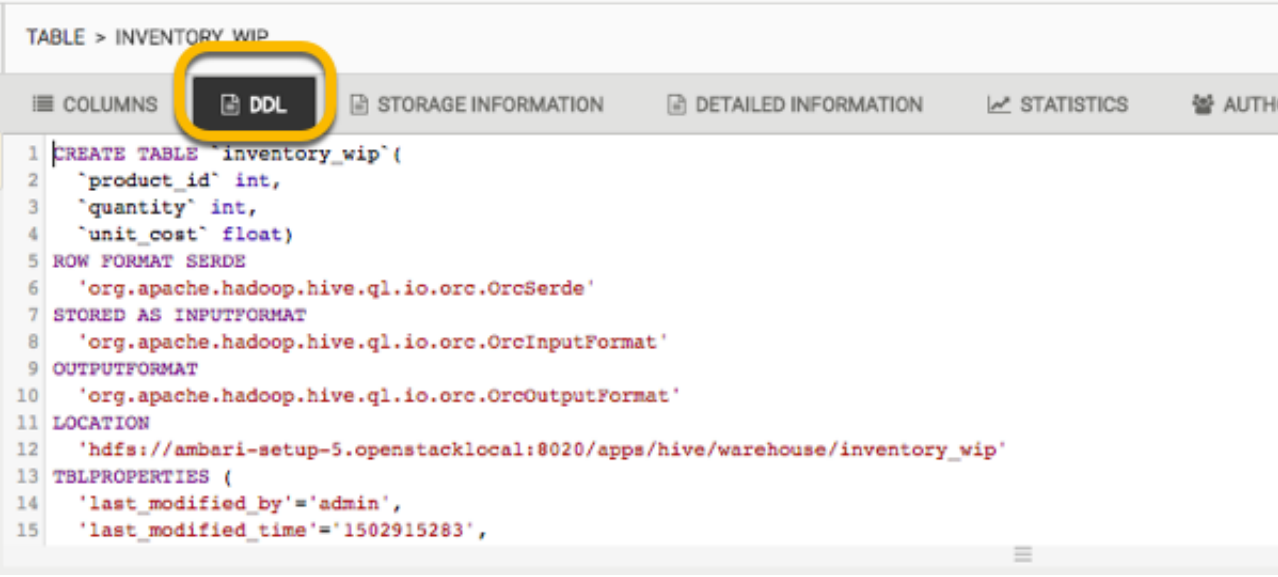
10. Click the **ADVANCED** subtab. Click the **Transactional** checkbox and enter your value for the **Number of buckets** field.



11. Click the **+Create** button toward the bottom of the Ambari window.

12. (Optional) Click the **DDL** subtab to view the SQL of the table so that you can copy it, use it as a template for other tables, and export it as a script. This can help with

understanding and possibly reusing the DDL in a production environment where you do not want to use a GUI tool, such as in a Beeline CLI.



2.2.4.4. SQL Standard-Based Authorization with GRANT and REVOKE SQL Statements

Secure SQL standard-based authorization using the GRANT and REVOKE SQL statements is supported in Hive 0.13 and later. Hive provides three authorization models: SQL standard-based authorization, storage-based authorization, and default Hive authorization.
In addition, Ranger provides centralized management of authorization for all HDP components. Use the following procedure to manually enable standard SQL authorization:

Note

This procedure is unnecessary if your Hive administrator installed Hive using Ambari.

1. Set the following configuration parameters in the `hive-site.xml` file:

Table 2.5. Configuration Parameters for Standard SQL Authorization

Configuration Parameter	Required Value
`hive.server2.enable.doAs`	`false`
`hive.users.in.admin.role`	Comma-separated list of users granted the administrator role.

2. Start HiveServer2 with the following command-line options:

Table 2.6. HiveServer2 Command-Line Options

Command-Line Option	Required Value
`-hiveconf hive.security.authorization.manager`	`org.apache.hadoop.hive.ql.security.authorization.MetaStoreAuthzAPIAuthorizerEmbedOnly`

Command-Line Option	Required Value
`-hiveconf hive.security.authorization.enabled`	`true`
`-hiveconf hive.security.authenticator.manager`	`org.apache.hadoop.hive.ql.security.SessionStateUserAuthenticator`
`-hiveconf hive.metastore.uris`	`''`(a space inside single quotation marks)

Note

Administrators must also specify a storage-based authorization manager for Hadoop clusters that also use storage-based authorization. The `hive.security.authorization.manager` configuration property allows multiple authorization managers in comma-delimited format, so the correct value in this case is:

```
hive.security.authorization.manager=org.apache.hadoop.hive.ql.
security.authorization.StorageBasedAuthorizationProvider,

org.apache.hadoop.hive.ql.security.authorization.
MetaStoreAuthzAPIAuthorizerEmbedOnly
```

2.2.4.5. Subqueries

Hive supports subqueries in FROM clauses and in WHERE clauses of SQL statements. A subquery is a SQL expression that is evaluated and returns a result set. Then that result set is used to evaluate the parent query. The parent query is the outer query that contains the child subquery. Subqueries in WHERE clauses are supported in Hive 0.13 and later. The following example shows a subquery inserted into a WHERE clause:

```
SELECT state, net_payments
FROM transfer_payments
WHERE transfer_payments.year IN (SELECT year FROM us_census);
```

No configuration is required to enable execution of subqueries in Hive. The feature is available by default. However, several restrictions exist for the use of subqueries in WHERE clauses.

Understanding Subqueries in SQL

SQL adheres to syntax rules like any programming language. The syntax governing the use of subqueries in WHERE clauses in SQL depends on the following concepts:

- **Query Predicates and Predicate Operators**

  A *predicate* in SQL is a condition that evaluates to a Boolean value. For example, the predicate in the preceeding example returns true for a row of the `transfer_payments` table if at least one row exists in the `us_census` table with the same year as the `transfer_payments` row. The predicate starts with the first WHERE keyword.

  ```
  ... WHERE transfer_payments.year IN (SELECT year FROM us_census);
  ```

  A SQL predicate in a subquery must also contain a predicate operator. *Predicate operators* specify the relationship tested in a predicate query. For example, the predicate operator in the above example is the IN keyword.

- **Aggregrated and Correlated Queries**

  Aggregated queries combine one or more aggregate functions, such as AVG, SUM, and MAX, with the GROUP BY statement to group query results by one or more table columns. In the following example, the AVG aggregate function returns the average salary of all employees in the engineering department grouped by year:

```
SELECT year, AVG(salary)
FROM Employees
WHERE department = 'engineering' GROUP BY year
```

  Note

  The GROUP BY statement may be either explicit or implicit.

  Correlated queries contain a query predicate with the equals (=) operator. One side of the operator must reference at least one column from the parent query and the other side must reference at least one column from the subquery. The following query is a revised and correlated version of the example query that is shown at the beginning of this section. It is a correlated query because one side of the equals predicate operator in the subquery references the `state` column in the `transfer_payments` table in the parent query and the other side of the operator references the `state` column in the `us_census` table.

```
SELECT state, net_payments
FROM transfer_payments
WHERE EXISTS
   (SELECT year
   FROM us_census
   WHERE transfer_payments.state = us_census.state);
```

  In contrast, an *uncorrelated query* does not reference any columns in the parent query.

- **Conjuncts and Disjuncts**

  A *conjunct* is equivalent to the AND condition, while a *disjunct* is the equivalent of the OR condition. The following subquery contains a conjunct:

```
... WHERE transfer_payments.year = "2010" AND us_census.state = "california"
```

  The following subquery contains a disjunct:

```
... WHERE transfer_payments.year = "2010" OR us_census.state = "california"
```

Restrictions on Subqueries in WHERE Clauses

Subqueries in WHERE clauses have the following limitations:

- Subqueries must appear on the right hand side of an expression.
- Nested subqueries are not supported.
- Only one subquery expression is allowed for a single query.
- Subquery predicates must appear as top level conjuncts.

- Subqueries support four logical operators in query predicates: IN, NOT IN, EXISTS, and NOT EXISTS.
- The IN and NOT IN logical operators may select only one column in a WHERE clause subquery.
- The EXISTS and NOT EXISTS operators must have at least one correlated predicate.
- The left side of a subquery must qualify all references to table columns.
- References to columns in the parent query are allowed only in the WHERE clause of the subquery.
- Subquery predicates that reference a column in a parent query must use the equals (=) predicate operator.
- Subquery predicates may not refer only to columns in the parent query.
- Correlated subqueries with an implied GROUP BY statement may return only one row.
- All unqualified references to columns in a subquery must resolve to tables in the subquery.
- Correlated subqueries cannot contain windowing clauses.

2.2.4.6. Common Table Expressions

A common table expression (CTE) is a set of query results obtained from a simple query specified within a WITH clause and which immediately preceeds a SELECT or INSERT keyword. A CTE exists only within the scope of a single SQL statement. One or more CTEs can be used with the following SQL statements:

- SELECT
- INSERT
- CREATE TABLE AS SELECT
- CREATE VIEW AS SELECT

The following example demonstrates the use of `q1` as a CTE in a SELECT statement:

```
WITH q1 AS (SELECT key from src where key = '5')
   SELECT * from q1;
```

The following example demonstrates the use of `q1` as a CTE in an INSERT statement:

```
CREATE TABLE s1 LIKE src;
   WITH q1 AS (SELECT key, value FROM src WHERE key = '5')
   FROM q1 INSERT OVERWRITE TABLE s1 SELECT *;
```

The following example demonstrates the use of `ql` as a CTE in a CREATE TABLE AS SELECT clause:

```
CREATE TABLE s2 AS WITH q1 AS (SELECT key FROM src WHERE key = '4')
   SELECT * FROM q1;
```

The following example demonstrates the use of q1 as a CTE in a CREATE TABLE AS VIEW clause:

```
CREATE VIEW v1 AS WITH q1 AS (SELECT key FROM src WHERE key='5')
   SELECT * from q1;
```

CTEs are available by default in Hive 0.13. Hive administrators do not need to perform any configuration to enable them.

Limitations of Common Table Expressions

- Recursive queries are not supported.
- The WITH clause is not supported within subquery blocks.

2.2.4.7. Quoted Identifiers in Column Names

Quoted identifiers in the names of table columns are supported in Hive 0.13 and later. An *identifier* in SQL is a sequence of alphanumeric and underscore (_) characters surrounded by backtick (`) characters. Quoted identifiers in Hive are case-insensitive. In the following example, `x+y` and `a?b` are valid column names for a new table.

```
CREATE TABLE test (`x+y` String, `a?b` String);
```

Quoted identifiers can be used anywhere a column name is expected, including table partitions and buckets:

```
CREATE TABLE partition_date-1 (key string, value string)
PARTITIONED BY (`dt+x` date, region int);

CREATE TABLE bucket_test(`key?1` string, value string)
CLUSTERED BY (`key?1`) into 5 buckets;
```

Note

Use a backtick character to escape a backtick character (``).

Enabling Quoted Identifiers

Set the `hive.support.quoted.identifiers` configuration parameter to `column` in the `hive-site.xml` file to enable quoted identifiers in SQL column names. For Hive 0.13, the valid values are `none` and `column`.

```
hive.support.quoted.identifiers = column
```

2.2.4.8. CHAR Data Type Support

Versions 0.13 and later support the CHAR data type. This data type simplifies the process of migrating data from other databases. Hive ignores trailing whitespace characters for the CHAR data type. However, there is no consensus among database vendors on the handling

of trailing whitespaces. Before you perform a data migration to Hive, consult the following table to avoid unexpected behavior with values for CHAR, VARCHAR, and STRING data types.

The following table describes how several types of databases treat trailing whitespaces for the CHAR, VARCHAR, and STRING data types:

Table 2.7. Trailing Whitespace Characters on Various Databases

Data Type	Hive	Oracle	SQL Server	MySQL	Teradata
CHAR	Ignore	Ignore	Ignore	Ignore	Ignore
VARCHAR	Compare	Compare	Configurable	Ignore	Ignore
STRING	Compare	N/A	N/A	N/A	N/A

2.2.5. Streaming Data Ingestion

Note

If you have questions regarding this feature, contact Support by logging a case on our Hortonworks Support Portal at http://support.hortonworks.com.

Limitations

Hive 0.13 and 0.14 have the following limitations to ingesting streaming data:

- Only ORC files are supported
- Destination tables must be bucketed
- Apache Flume or Apache Storm may be used as the streaming source

2.2.6. Query Vectorization

Vectorization allows Hive to process a batch of rows together instead of processing one row at a time. Each batch is usually an array of primitive types. Operations are performed on the entire column vector, which improves the instruction pipelines and cache usage. HIVE-4160 has the design document for vectorization and tracks the implementation of many subtasks.

Enable Vectorization in Hive

To enable vectorization, set this configuration parameter:

```
hive.vectorized.execution.enabled=true
```

When vectorization is enabled, Hive examines the query and the data to determine whether vectorization can be supported. If it cannot be supported, Hive will execute the query with vectorization turned off.

Log Information about Vectorized Execution of Queries

The Hive client will log, at the `info` level, whether a query's execution is being vectorized. More detailed logs are printed at the `debug` level.

The client logs can also be configured to show up on the console.

Supported Functionality

The current implementation supports only single table read-only queries. DDL queries or DML queries are not supported.

The supported operators are **selection**, **filter** and **group by**.

Partitioned tables are supported.

These data types are supported:

- tinyint
- smallint
- int
- bigint
- date
- boolean
- float
- double
- timestamp
- string
- char
- varchar
- binary

These expressions are supported:

- Comparison: >, >=, <, <=, =, !=
- Arithmetic: plus, minus, multiply, divide, modulo
- Logical: AND, OR
- Aggregates: sum, avg, count, min, max

Only the ORC file format is supported in the current implementation.

2.2.7. Beeline versus Hive CLI

HDP supports two Hive clients: the Hive CLI and Beeline. The primary difference between the two involves how the clients connect to Hive.

- The Hive CLI, which connects directly to HDFS and the Hive Metastore, and can be used only on a host with access to those services.
- Beeline, which connects to HiveServer2 and requires access to only one .jar file: `hive-jdbc-<version>-standalone.jar`.

Hortonworks recommends using HiveServer2 and a JDBC client (such as Beeline) as the primary way to access Hive. This approach uses SQL standard-based authorization or Ranger-based authorization. However, some users may wish to access Hive data from other applications, such as Pig. For these use cases, use the Hive CLI and storage-based authorization.

Beeline Operating Modes and HiveServer2 Transport Modes

Beeline supports the following modes of operation:

Table 2.8. Beeline Modes of Operation

Operating Mode	Description
Embedded	The Beeline client and the Hive installation both reside on the same host machine. No TCP connectivity is required.
Remote	Use remote mode to support multiple, concurrent clients executing queries against the same remote Hive installation. Remote transport mode supports authentication with LDAP and Kerberos. It also supports encryption with SSL. TCP connectivity is required.

Administrators may start HiveServer2 in one of the following transport modes:

Table 2.9. HiveServer2 Transport Modes

Transport Mode	Description
TCP	HiveServer2 uses TCP transport for sending and receiving Thrift RPC messages.
HTTP	HiveServer2 uses HTTP transport for sending and receiving Thrift RPC messages.

While running in TCP transport mode, HiveServer2 supports the following authentication schemes:

Table 2.10. Authentication Schemes with TCP Transport Mode

Authentication Scheme	Description
Kerberos	A network authentication protocol which operates that uses the concept of 'tickets' to allow nodes in a network to securely identify themselves. Administrators must specify `hive.server2.authentication=kerberos`, `hive.server2.authentication.kerberos.principal = hive/_HOST@YOUR-REALM.COM`, and `hive.server2.authentication.kerberos.keytab = /etc/hive/conf/hive.keytab` in the hive-site.xml configuration file to use this authentication scheme.

Authentication Scheme	Description
LDAP	The Lightweight Directory Access Protocol, an application-layer protocol that uses the concept of 'directory services' to share information across a network. Administrators must specify hive.server2.authentication=ldap in the hive-site.xml configuration file to use this type of authentication.
PAM	Pluggable Authentication Modules, or PAM, allow administrators to integrate multiple authentication schemes into a single API. Administrators must specify hive.server2.authentication=pam in the hive-site.xml configuration file to use this authentication scheme.
Custom	Authentication provided by a custom implementation of the org.apache.hive.service.auth.PasswdAuthenticationProvider interface. The implementing class must be available in the classpath for HiveServer2 and its name provided as the value of the hive.server2.custom.authentication.class property in the hive-site.xml configuration property file.
None	The Beeline client performs no authentication with HiveServer2. Administrators must specify hive.server2.authentication=none in the hive-site.xml configuration file to use this authentication scheme.

The next section describes the connection strings used to connect to HiveServer2 for all possible combinations of these modes, as well as the connection string required to connect to HiveServer2 in a secure cluster.

Connecting to Hive with Beeline

The following examples demonstrate how to use Beeline to connect to Hive for all possible variations of these modes.

Embedded Client

Use the following syntax to connect to Hive from Beeline in embedded mode:

```
!connect jdbc:hive2://
```

Remote Client with HiveServer2 TCP Transport Mode and SASL Authentication

While running in TCP transport mode, HiveServer2 uses the Java Simple Authentication and Security Layer (SASL) protocol to establish a security layer between the client and server. Use the following syntax to connect to HiveServer2 in TCP mode from a remote Beeline client:

```
!connect jdbc:hive2://<host>:<port>/<db>
```

The default port for HiveServer2 in TCP mode is `10000`, and `db` is the name of the database to which you want to connect.

Remote Client with HiveServer2 HTTP Transport Mode

Use the following syntax to connect to HiveServer2 in HTTP mode from a remote Beeline client:

```
!connect jdbc:hive2://<host>:<port>/<db>;hive.server2.transport.mode=
http;hive.server2.thrift.http.path=<http_endpoint>
```

Note

The value for <http_endpoint> can be found in the `hive.server2.thrift.http.path` property in the `hive-site.xml` file. If it is not listed there, use the default value `cliservice`.

In an environment that is secured by Kerberos, use the following syntax to connect to HiveServer2 in HTTP mode from a remote Beeline client:

```
!connect jdbc:hive2://<host>:<port>/<db>;hive.server2.transport.mode=
http;hive.server2.thrift.http.path=<http_endpoint>;principal=hive/HOST@REALM
```

Remote Client with HiveServer2 in Secure Cluster

Use the following syntax to connect to HiveServer2 in a secure cluster from a remote Beeline client:

```
!connect jdbc:hive2://<host>:<port>/<db>;principal=
<Server_Principal_of_HiveServer2>
```

Note

The Beeline client must have a valid Kerberos ticket in the ticket cache before attempting to connect.

2.2.8. Hive JDBC and ODBC Drivers

Hortonworks provides Hive JDBC and ODBC drivers that let you connect to popular Business Intelligence (BI) tools to query, analyze and visualize data stored within the Hortonworks Data Platform.

For information about using the Hive ODBC drivers and to download a driver, on the Hortonworks Downloads page, click Addons.

Current Hive JDBC client jars can be found on one of the edge nodes in your cluster at `/usr/hdp/current/hive-client/lib/hive-jdbc.jar` after you have installed HDP, or you can download them from Hive JDBC driver archive.

Note

Some HiveServer2 clients may need to run on a host outside of the Hadoop cluster. These clients require access to the following .jar files to successfully use the Hive JDBC driver in both HTTP and HTTPS modes: `hive-jdbc-<version>-standalone.jar`, `hadoop-common.jar`, and `hadoop-auth.jar`.

JDBC URLs have the following format:

```
jdbc:hive2://<host>:<port>/<dbName>;<sessionConfs>?<hiveConfs>#<hiveVars>
```

JDBC Parameter	Description
`host`	The cluster node hosting HiveServer2.

JDBC Parameter	Description
`port`	The port number to which HiveServer2 listens.
`dbName`	The name of the Hive database to run the query against.
`sessionConfs`	Optional configuration parameters for the JDBC/ODBC driver in the following format: `<key1>=<value1>;<key2>=<key2>...;`
`hiveConfs`	Optional configuration parameters for Hive on the server in the following format: `<key1>=<value1>;<key2>=<key2>; ...` The configurations last for the duration of the user session.
`hiveVars`	Optional configuration parameters for Hive variables in the following format: `<key1>=<value1>;<key2>=<key2>; ...` The configurations last for the duration of the user session.

The specific JDBC connection URL for a HiveServer2 client depends on several factors:

- How is HiveServer2 deployed on the cluster?
- What type of transport does HiveServer2 use?
- Does HiveServer2 use transport-layer security?
- Is HiveServer2 configured to authenticate users?

The rest of this section describes how to use session configuration variables to format the JDBC connection URLs for all these scenarios.

Embedded and Remote Modes

In embedded mode, HiveServer2 runs within the Hive client rather than in a separate process. No host or port number is necessary for the JDBC connection. In remote mode, HiveServer2 runs as a separate daemon on a specified host and port, and the JDBC client and HiveServer2 interact using remote procedure calls with the Thrift protocol.

Embedded Mode

```
jdbc:hive2://
```

Remote Mode

```
jdbc:hive2://<host>:<port>/<dbName>;<sessionConfs>?<hiveConfs>#<hiveVars>
```

Note

The rest of the example JDBC connection URLs in this topic are valid only for HiveServer2 configured in remote mode.

TCP and HTTP Transport

The JDBC client and HiveServer2 can use either HTTP or TCP-based transport to exchange RPC messages. Specify the transport used by HiveServer2 with the `transportMode` and `httpPath` session configuration variables. The default transport is TCP.

transportMode Variable Value	Description
`http`	Connect to HiveServer2 using HTTP transport.
`binary`	Connect to HiveServer2 using TCP transport.

HTTP Transport

```
jdbc:hive2://<host>:<port>/<dbName>;transportMode=http;httpPath=
<http_endpoint>;<otherSessionConfs>?<hiveConfs>#<hiveVars>
```

Note

The JDBC driver assumes a value of `cliservice` if the `httpPath` configuration variable is not specified.

TCP Transport

```
jdbc:hive2://<host>:<port>/<dbName>;<otherSessionConfs>?
<hiveConfs>#<hiveVars>
```

Because the default transport is TCP, there is no need to specify `transportMode=binary` if TCP transport is desired.

User Authentication

HiveServer2 supports Kerberos, LDAP, Pluggable Authentication Modules (PAM), and custom plugins for authenticating the JDBC user connecting to HiveServer2. The format of the JDBC connection URL for authentication with Kerberos differs from the format for other authentication models.

User Authentication Variable	Description
`principal`	A string that uniquely identifies a Kerberos user.
`saslQop`	Quality of protection for the SASL framework. The level of quality is negotiated between the client and server during authentication. Used by Kerberos authentication with TCP transport.
`user`	Username for non-Kerberos authentication model.
`password`	Password for non-Kerberos authentication model.

Kerberos Authentication

```
jdbc:hive2://<host>:<port>/<dbName>;principal=
<HiveServer2_kerberos_principal>;<otherSessionConfs>?<hiveConfs>#<hiveVars>
```

Kerberos Authentication with Sasl QOP

```
jdbc:hive2://<host>:<port>/<dbName>;principal=
<HiveServer2_kerberos_principal>;saslQop=<qop_value>;<otherSessionConfs>?
<hiveConfs>#<hiveVars>
```

Non-Kerberos Authentication

```
jdbc:hive2://<host>:<port>/<dbName>;user=<username>;password=
<password>;<otherSessionConfs>?<hiveConfs>#<hiveVars>
```

Transport Layer Security

HiveServer2 supports SSL and Sasl QOP for transport-layer security. The format of the JDBC connection URL for SSL differs from the format used by Sasl QOP.

SSL Variable	Description
`ssl`	Specifies whether to use SSL
`sslTrustStore`	The path to the SSL TrustStore.
`trustStorePassword`	The password to the SSL TrustStore.

```
jdbc:hive2://<host>:<port>/<dbName>;ssl=true;sslTrustStore=
<ssl_truststore_path>;trustStorePassword=
<truststore_password>;<otherSessionConfs>?<hiveConfs>#<hiveVars>
```

When using TCP for transport and Kerberos for security, HiveServer2 uses Sasl QOP for encryption rather than SSL.

Sasl QOP Variable	Description
`principal`	A string that uniquely identifies a Kerberos user.
`saslQop`	The level of protection desired. For authentication, checksum, and encryption, specify `auth-conf`. The other valid values do not provide encryption.

```
jdbc:hive2://<host>:<port>/<dbName>;principal=
<HiveServer2_kerberos_principal>;saslQop=auth-conf;<otherSessionConfs>?
<hiveConfs>#<hiveVars>
```

2.3. Moving Data into Apache Hive

There are multiple methods of moving data into Hive. How you move the data into Hive depends on the source format of the data and the target data format that is required. Generally, ORC is the preferred target data format because of the performance enhancements that it provides.

The following methods are most commonly used:

Table 2.11. Most Common Methods to Move Data into Hive

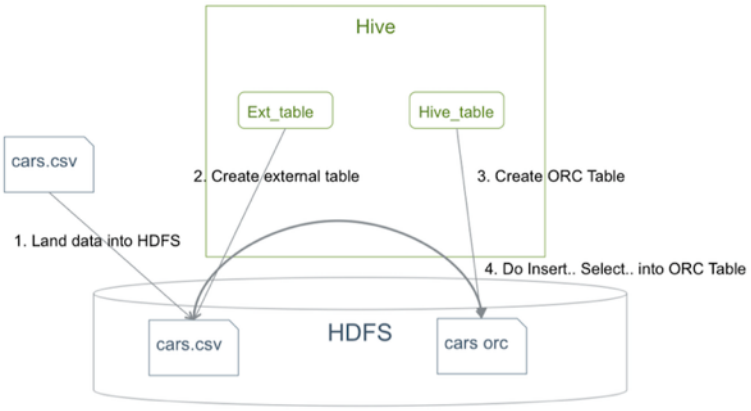
Source of Data	Target Data Format in Hive	Method Description
ETL for legacy systems	ORC file format	1. Move data into HDFS. 2. Use an external table to move data from HDFS to Hive. 3. Then use Hive to convert the data to the ORC file format.
Operational SQL database	ORC file format	1. Use Sqoop to import the data from the SQL database into Hive. 2. Then use Hive to convert the data to the ORC file format.

Source of Data	Target Data Format in Hive	Method Description
Streaming source that is "append only"	ORC file format	1. Write directly to the ORC file format using the Hive Streaming feature.

2.3.1. Using an External Table

This is the most common way to move data into Hive when the ORC file format is required as the target data format. Then Hive can be used to perform a fast parallel and distributed conversion of your data into ORC. The process is shown in the following diagram:

Figure 2.1. Example: Moving .CSV Data into Hive



Moving .CSV Data into Hive

The following steps describe moving .CSV data into Hive using the method illustrated in the above diagram with command-line operations.

1. **Move .CSV data into HDFS**:

   a. The following is a .CSV file which contains a header line that describes the fields and subsequent lines that contain the data:

   ```
   [<username>@cn105-10 ~]$ head cars.csv
   Name,Miles_per_Gallon,Cylinders,Displacement,Horsepower,Weight_in_lbs,
   Acceleration,Year,Origin
   "chevrolet chevelle malibu",18,8,307,130,3504,12,1970-01-01,A
   "buick skylark 320",15,8,350,165,3693,11.5,1970-01-01,A
   "plymouth satellite",18,8,318,150,3436,11,1970-01-01,A
   "amc rebel sst",16,8,304,150,3433,12,1970-01-01,A
   "ford torino",17,8,302,140,3449,10.5,1970-01-01,A
   ...
   [<username>@cn105-10 ~]$
   ```

   <username> is the user who is performing the operation. To test this example, run with a user from your environment.

   b. First, use the following command to remove the header line from the file because it is not part of the data for the table:

   ```
   [<username>@cn105-10 ~]$ sed -i 1d cars.csv
   ```

c. Move the data to HDFS:

```
[<username>@cn105-10 ~]$ hdfs dfs -copyFromLocal cars.csv /user/
<username>/visdata
[<username>@cn105-10 ~]$ hdfs dfs -ls /user/<username>/visdata
Found 1 items
-rwxrwxrwx   3 <username> hdfs      22100 2015-08-12 16:16 /user/
<username>/visdata/cars.csv
```

2. **Create an external table**.

   An *external table* is a table for which Hive does not manage storage. If you delete an external table, only the definition in Hive is deleted. The data remains. An *internal table* is a table that Hive manages. If you delete an internal table, both the definition in Hive *and* the data are deleted.

   The following command creates an external table:

```
CREATE EXTERNAL TABLE IF NOT EXISTS Cars(
        Name STRING,
        Miles_per_Gallon INT,
        Cylinders INT,
        Displacement INT,
        Horsepower INT,
        Weight_in_lbs INT,
        Acceleration DECIMAL,
        Year DATE,
        Origin CHAR(1))
    COMMENT 'Data about cars from a public database'
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ','
    STORED AS TEXTFILE
    location '/user/<username>/visdata';
```

3. **Create the ORC table**.

   Now, create a table that is managed by Hive with the following command:

```
CREATE TABLE IF NOT EXISTS mycars(
        Name STRING,
        Miles_per_Gallon INT,
        Cylinders INT,
        Displacement INT,
        Horsepower INT,
        Weight_in_lbs INT,
        Acceleration DECIMAL,
        Year DATE,
        Origin CHAR(1))
    COMMENT 'Data about cars from a public database'
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ','
    STORED AS ORC;
```

4. **Insert the data from the external table to the Hive ORC table**.

   Now, use an SQL statement to move the data from the external table that you created in Step 2 to the Hive-managed ORC table that you created in Step 3:

```
INSERT OVERWRITE TABLE mycars SELECT * FROM cars;
```

Note

Using Hive to convert an external table into an ORC file format is very efficient because the conversion is a parallel and distributed action, and no standalone ORC conversion tool is necessary.

5. **Verify that you imported the data into the ORC-formatted table correctly**:

```
hive> select * from mycars limit 3;
OK
"chevrolet chevelle malibu" 18 8 307 130 3504 12 1970-01-01 A
"buick skylark 320" 15 8 350 165 3693 12 1970-01-01 A
"plymouth satellite" 18 8 318 150 3436 11 1970-01-01 A
Time taken: 0.144 seconds, Fetched: 3 row(s)
```

2.3.2. Using Sqoop

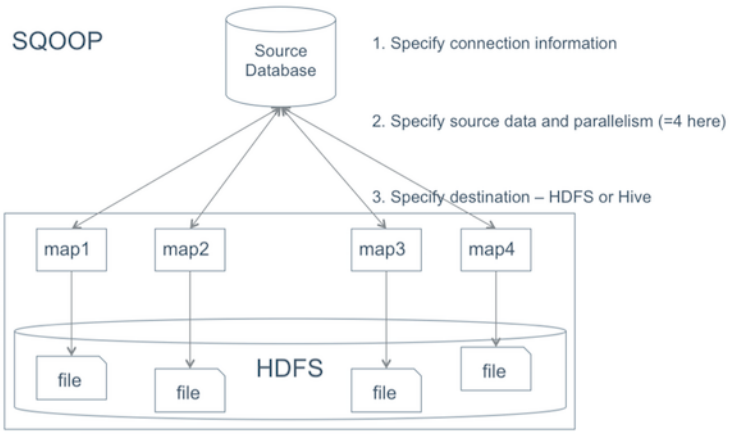
Sqoop is a tool that enables you to bulk import and export data from a database. You can use Sqoop to import data into HDFS or directly into Hive. However, Sqoop can only import data into Hive as a text file or as a SequenceFile. To use the ORC file format, you must use a two-phase approach: first use Sqoop to move the data into HDFS, and then use Hive to convert the data into the ORC file format as described in the above Steps 3 and 4 of "Moving Data from HDFS to Hive Using an External Table."

For more information on using Sqoop, refer to Using Apache Sqoop to Transfer Bulk Data.

A detailed Sqoop user guide is also available on the Apache web site here.

The process for using Sqoop to move data into Hive is shown in the following diagram:

Figure 2.2. Using Sqoop to Move Data into Hive



Moving Data into Hive Using Sqoop

1. **Specify the source connection information**.

First, you must specify the:

- database URI (`db.foo.com` in the following example)
- database name (`bar`)
- connection protocol (`jdbc:mysql:`)

For this example, use the following command:

```
sqoop import --connect jdbc:mysql://db.foo.com/bar --table EMPLOYEES
```

If the source database requires credentials, such as a username and password, you can enter the password on the command line or specify a file where the password is stored.

For example:

- Enter the password on the command line:

```
sqoop import --connect jdbc:mysql://db.foo.com/bar --table EMPLOYEES --
username <username> -P
Enter password: (hidden)
```

- Specify a file where the password is stored:

```
sqoop import --connect jdbc:mysql://db.foo.com/bar --table EMPLOYEES --
username <username> --password-file ${user.home}/.password
```

Sqoop is tested to work with Connector/J 5.1. If you have upgraded to Connector/J 8.0, and want to use the zeroDateTimeBehavior to handle values of '0000-00-00\' in DATE columns, explicitly specify `zeroDateTimeBehavior=CONVERT_TO_NULL` in the connection string. For example,

```
jdbc:mysql://<MySQL host>/<DB>?zeroDateTimeBehavior=CONVERT_TO_NULL
```

More connection options are described in the Sqoop User Guide on the Apache web site.

2. **Specify the data and the parallelism for import**:

   a. Specify the data simply.

   Sqoop provides flexibility to specify exactly the data you want to import from the source system:

   - Import an **entire table**:

```
sqoop import --connect jdbc:mysql://db.foo.com/bar --table EMPLOYEES
```

   - Import a **subset of the columns** from a table:

```
sqoop import --connect jdbc:mysql://db.foo.com/bar --table EMPLOYEES --
columns "employee_id,first_name,last_name,job_title"
```

   - Import only the **latest records** by specifying them with a WHERE clause and then that they be appended to an existing table:

```
sqoop import --connect jdbc:mysql://db.foo.com/bar --table EMPLOYEES
 --where "start_date > '2010-01-01'"

sqoop import --connect jdbc:mysql://db.foo.com/bar --table EMPLOYEES --
where "id > 100000" --target-dir /incremental_dataset --append
```

You can also use a free-form SQL statement.

b. Specify parallelism.

There are three options for specifying *write parallelism* (number of map tasks):

- Explicitly set the number of mappers using `--num-mappers`. Sqoop evenly splits the primary key range of the source table:

```
sqoop import --connect jdbc:mysql://db.foo.com/bar --table EMPLOYEES --
num-mappers 8
```

In this scenario, the source table must have a primary key.

- Provide an alternate split key using `--split-by`. This evenly splits the data using the alternate split key instead of a primary key:

```
sqoop import --connect jdbc:mysql://db.foo.com/bar --table EMPLOYEES --
split-by dept_id
```

This method is useful if primary keys are not evenly distributed.

- When there is not split key or primary key, the data import must be sequential. Specify a single mapper by using `--num-mappers 1` or `--autoreset-to-one-mapper`.

c. Specify the data using a query.

Instead of specifying a particular table or columns, you can specify the date with a query. You can use one of the following options:

- Explicitly specify a *split-by column* using `--split-by` and put `$ CONDITIONS` that Sqoop replaces with range conditions based on the split-by key. This method requires a target directory:

```
sqoop import --query 'SELECT a.*, b.* FROM a JOIN b on (a.id == b.id)
WHERE $CONDITIONS' --split-by a.id --target-dir /user/foo/joinresults
```

- Use sequential import if you cannot specify a split-by column:

```
sqoop import --query 'SELECT a.*, b.* FROM a JOIN b on (a.id == b.id)
WHERE $CONDITIONS' -m 1 --target-dir /user/foo/joinresults
```

To try a sample query without importing data, use the `eval` option to print the results to the command prompt:

```
sqoop eval --connect jdbc:mysql://db.foo.com/bar --query "SELECT * FROM
 employees LIMIT 10"
```

3. **Specify the destination for the data: HDFS or Hive**.

Here is an example of specifying the HDFS target directory:

```
sqoop import --query 'SELECT a.*, b.* FROM a JOIN b on (a.id == b.id)
WHERE $CONDITIONS' --split-by a.id --target-dir /user/foo/joinresults
```

If you can add text data into your Hive table, you can specify that the data be directly added to Hive. Using `--hive-import` is the primary method to add text data directly to Hive:

```
sqoop import --connect jdbc:mysql://db.foo.com/corp --table EMPLOYEES --
hive-import
```

This method creates a metastore schema after storing the text data in HDFS.

If you have already moved data into HDFS and want to add a schema, use the `create-hive-table` Sqoop command:

```
sqoop create-hive-table (generic-args) (create-hive-table-args)
```

Additional options for importing data into Hive with Sqoop:

Table 2.12. Sqoop Command Options for Importing Data into Hive

Sqoop Command Option	Description
`--hive-home <directory>`	Overrides `$HIVE_HOME`.
`--hive-import`	Imports tables into Hive using Hive's default delimiters if none are explicitly set.
`--hive-overwrite`	Overwrites existing data in the Hive table.
`--create-hive-table`	Creates a hive table during the operation. If this option is set and the Hive table already exists, the job will fail. Set to `false` by default.
`--hive-table <table_name>`	Specifies the table name to use when importing data into Hive.
`--hive-drop-import-delims`	Drops the delimiters `\n`, `\r`, and `\01` from string fields when importing data into Hive.
`--hive-delims-replacement`	Replaces the delimiters `\n`, `\r`, and `\01` from strings fields with a user-defined string when importing data into Hive.
`--hive-partition-key`	Specifies the name of the Hive field on which a sharded database is partitioned.
`--hive-partition-value <value>`	A string value that specifies the partition key for data imported into Hive.
`--map-column-hive <map>`	Overrides the default mapping from SQL type to Hive type for configured columns.

2.3.3. Incrementally Updating a Table

It is common to perform a one-time ingestion of data from an operational database to Hive and then require incremental updates periodically. You can perform periodic updates as described in this section.

Note

This procedure requires change data capture from the operational database that has a primary key and modified date field where you pulled the records from since the last update.

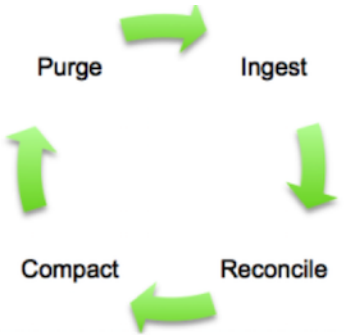
Overview

This procedure combines the techniques that are described in the sections "Moving Data from HDFS to Hive Using an External Table" and "Using Sqoop to Move Data into Hive."

Use the following steps to incrementally update Hive tables from operational database systems:

1. **Ingest:** Complete data movement from the operational database (base_table) followed by change or update of changed records only (incremental_table).
2. **Reconcile:** Create a single view of the base table and change records (reconcile_view) to reflect the latest record set.
3. **Compact:** Create a reporting table (reporting_table) from the reconciled view.
4. **Purge:** Replace the base table with the reporting table contents and delete any previously processed change records before the next data ingestion cycle, which is shown in the following diagram.

Figure 2.3. Data Ingestion Lifecycle



The base table is a Hive internal table, which was created during the first data ingestion. The incremental table is a Hive external table, which likely is created from .CSV data in HDFS. This external table contains the changes (INSERTs and UPDATEs) from the operational database since the last data ingestion.

Note

Generally, the table is partitioned and only the latest partition is updated, making this process more efficient.

Incrementally Updating Data in Hive

1. **Ingest the data.**
   a. Store the base table in the ORC format in Hive.

The first time that data is ingested, you must import the entire table from the source database. You can use Sqoop. The following example shows importing data from Teradata:

```
sqoop import --connect jdbc:teradata://{host name}/Database=retail
--connection-manager org.apache.sqoop.teradata.TeradataConnManager --
username dbc
--password dbc --table SOURCE_TBL --target-dir /user/hive/base_table -m 1
```

b. Store this data into an ORC-formatted table using the Steps 2 - 5 shown in "Moving Data from HDFS to Hive Using an External Table."

The base table definition after moving it from the external table to a Hive-managed table looks like the below example:

```
CREATE TABLE base_table (
        id STRING,
        field1 STRING,
        modified_date DATE)
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ','
    STORED AS ORC;
```

c. Store the incremental table as an external table in Hive.

It is more common to be importing incremental changes since the last time data was updated and then merging it. See the section "Using Sqoop to Move Data into Hive" for examples of importing data with Sqoop.

In the following example, `--check-column` is used to fetch records newer than `last_import_date`, which is the date of the last incremental data update:

```
sqoop import --connect jdbc:teradata://{host name}/Database=retail
--connection-manager org.apache.sqoop.teradata.TeradataConnManager
--username dbc --password dbc --table SOURCE_TBL --target-dir /user/hive/
incremental_table -m 1
--check-column modified_date --incremental lastmodified --last-value
 {last_import_date}
```

You can also use `--query` to perform the same operation:

```
sqoop import --connect jdbc:teradata://{host name}/Database=retail
--connection-manager org.apache.sqoop.teradata.TeradataConnManager --
username dbc
--password dbc --target-dir /user/hive/incremental_table -m 1
--query 'select * from SOURCE_TBL where modified_date >
 {last_import_date} AND $CONDITIONS'
```

d. After the incremental table data is moved into HDFS using Sqoop, you can define an external Hive table over it with the following command:

```
CREATE EXTERNAL TABLE incremental_table (
        id STRING,
        field1 STRING,
        modified_date DATE)
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ','
    STORED AS TEXTFILE
    location '/user/hive/incremental_table';
```

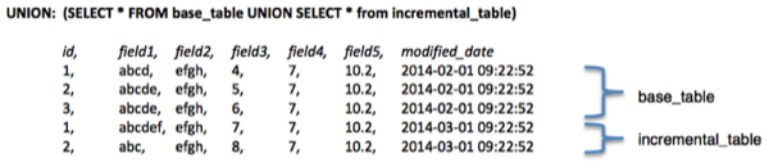
2. **Reconcile or merge the data.**

   Create a view that uses UNION ALL to merge the data and reconcile the base table records with the new records:

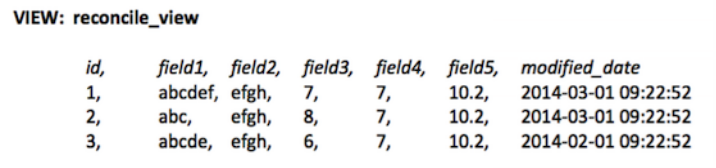
```
CREATE VIEW reconcile_view AS
SELECT t1.* FROM
    (SELECT * FROM base_table
     UNION ALL
     SELECT * from incremental_table) t1
JOIN
    (SELECT id, max(modified_date) max_modified FROM
        (SELECT * FROM base_table
         UNION ALL
         SELECT * from incremental_table)
     GROUP BY id) t2
ON t1.id = t2.id AND t1.modified_date = t2.max_modified;
```

   EXAMPLES:

   - **Figure 2.4. Dataset after the UNION ALL Command Is Run**



   - **Figure 2.5. Dataset in the View**



Note

In the `reconcile_view` only one record exists per primary key, which is shown in the `id` column. The values displayed in the `id` column correspond to the latest modification date that is displayed in the `modified_date` column.

3. **Compact the data.**

The view changes as soon as new data is introduced into the incremental table in HDFS (`/user/hive/incremental_table`, so create and store a copy of the view as a snapshot in time:

```
DROP TABLE reporting_table;
CREATE TABLE reporting_table AS
SELECT * FROM reconcile_view;
```

4. **Purge data.**

   a. After you have created a reporting table, clean up the incremental updates to ensure that the same data is not read twice:

   ```
   hadoop fs –rm –r /user/hive/incremental_table/*
   ```

   b. Move the data into the ORC format as the base table. Frequently, this involves a partition rather than the entire table:

   ```
   DROP TABLE base_table;
   CREATE TABLE base_table (
           id STRING,
           field1 STRING,
           modified_date DATE)
       ROW FORMAT DELIMITED
       FIELDS TERMINATED BY ','
       STORED AS ORC;

   INSERT OVERWRITE TABLE base_table SELECT * FROM reporting_table;
   ```

Handling Deletes

Deletes can be handled by adding a DELETE_DATE field in the tables:

```
CREATE VIEW reconcile_view AS
    SELECT t1.* FROM
        (SELECT * FROM base_table
         UNION
         SELECT * FROM incremental_table) t1
     JOIN
         (SELECT id, max(modified_date) max_modified FROM
             (SELECT * FROM base_table
              UNION
              SELECT * FROM incremental_table)
      GROUP BY id) t2
   ON t1.id = t2.id AND t1.modified_date = t2.max_modified
   AND t1.delete_date IS NULL;
```

Tip

You can automate the steps to incrementally update data in Hive by using Oozie. See "Using HDP for Workflow and Scheduling (Oozie)."

2.4. Queries on Data Stored in Remote Clusters

Some environments upgrade HDP Hive on a cluster but leave an older version on a different cluster. This setup includes the scenario where you have a compute-only cluster with a

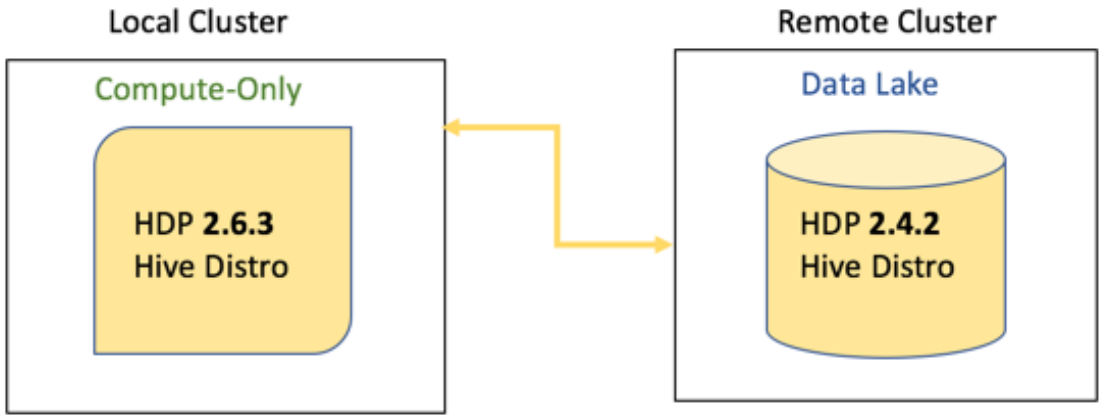
recent version of Hive that runs simultaneously with a data-lake cluster hosting an older Hive version. In this scenario, you might want to run queries using the resources of the latest HDP Hive installation on the older Hive installation that maintains the data.

Hive distributed in HDP version 2.6.3 supports functionality that lets you query remote clusters hosting HDP Hive version 2.4.3 data from a cluster that has the Hive distribution in HDP 2.6.3. This functionality includes both READ and WRITE privileges on the data.

2.4.1. Query Capability on Remote Clusters

Important

Hortonworks certification of the ability to query remote clusters as described in this section applies only to environments where the data lake is stored in a HDP 2.4.2 Hive data warehouse and the local cluster for computation has HDP 2.6.3 Hive.



The DDL statements to create an infrastructure where the local, compute-only cluster can query data on a remote cluster are documented on the LanguageManual DDL page of the Apache Hive wiki. The only additional requirement is that the LOCATION clause must point to the IP address of the active NameNode in the remote cluster.

Required Setup of Database Objects

Important

The SQL statements in this section are examples to illustrate possible ways of creating the objects with properties that support queries on remote clusters. Do not copy these SQL statements and paste them into your production system.

- An external table on the local cluster that exposes the Hive data tracked by an active NameNode and that specifies the IP address of the NameNode in the LOCATION clause of the SQL statement.

  Example of how to create an external table pointing to the active NameNode:

```
CREATE EXTERNAL TABLE orders_ext
(
  O_ORDERKEY int,
  O_CUSTKEY int,
  O_ORDERSTATUS char(1),
  O_TOTALPRICE decimal(15,2),
  O_ORDERDATE date,
  O_ORDERPRIORITY char(15),
  O_CLERK char(15),
  O_SHIPPRIORITY int,
  O_COMMENT string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'
LOCATION "hdfs://172.27.24.198:8020/tmp/tpch/data/orders";
```

- A Hive table on the remote cluster into which the data of the active NameNode is loaded.

  Example of how to create a Hive table and load it with data of a NameNode IP address:

```
CREATE TABLE orders_managed
(
  O_ORDERKEY int,
  O_CUSTKEY int,
  O_ORDERSTATUS char(1),
  O_TOTALPRICE decimal(15,2),
  O_ORDERDATE date,
  O_ORDERPRIORITY char(15),
  O_CLERK char(15),
  O_SHIPPRIORITY int,
  O_COMMENT string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|';

load data inpath "hdfs://172.27.24.198:8020/tmp/orders" into table
 orders_managed;
```

- Some environments also require temporary tables or views to manipulate a subset of the data or cache the results of a large query.

Next Steps

You can query the data on the remote cluster, including the ability to perform WRITE operations from the local cluster.

Examples of Supported Queries

```
CREATE TABLE orders_ctas AS SELECT * FROM orders_ext;
```

```
INSERT INTO orders_ctas SELECT * FROM orders_ext;
```

```
INSERT OVERWRITE TABLE orders_ctas SELECT * FROM orders_ext;
```

2.5. Configuring HiveServer2

HiveServer2 is a server interface that enables remote clients to execute queries against Hive and retrieve the results. This section describes how to configure HiveServer2 for transactions (ACID) and how to configure HiveServer2 for LDAP and for LDAP over SSL.

2.5.1. Configuring HiveServer2 for Transactions (ACID Support)

Hive supports transactions that adhere to traditional relational database ACID characteristics: atomicity, consistency, isolation, and durability. See the article about ACID characteristics on Wikipedia for more information.

Limitations

Currently, ACID support in Hive has the following limitations:

- `BEGIN`, `COMMIT`, and `ROLLBACK` are not yet supported.
- Only the ORC file format is supported.
- Transactions are configured to be off by default.
- Tables that use transactions must be bucketed. For a discussion of bucketed tables, see the Apache site.
- Hive ACID only supports Snapshot Isolation.
- Transactions only support auto-commit mode and may include exactly one SQL statement.
- ZooKeeper and in-memory lock managers are not compatible with transactions. See the Apache site for a discussion of how locks are stored for transactions.

To configure HiveServer2 for transactions:

Important

- Ensure that the `hive.txn.timeout` property is set to the same value in the `hive-site.xml` file for HiveServer2 that you configure in Step 1 below and the `hive-site.xml` file for the standalone Hive metastore that you configure in Step 2.
- The following listed properties are the minimum that are required to enable transaction support on HiveServer2. For additional information about configuring this feature and for information about additional configuration parameters, see Hive Transactions on the Apache web site.

1. Set the following parameters in the `hive-site.xml` file:

```
<property>
     <name>hive.support.concurrency</name>
     <value>true</value>
</property>

<property>
     <name>hive.txn.manager</name>
     <value>org.apache.hadoop.hive.ql.lockmgr.DbTxnManager</value>
</property>

<property>
     <name>name>hive.enforce.bucketing</name>
```

```
    <value>true</value>
</property>

<property>
    <name>hive.exec.dynamic.partition.mode</name>
    <value>nostrict</value>
</property>
```

2. Ensure that a standalone Hive metastore is running with the following parameters set in its `hive-site.xml` file:

```
<property>
    <name>hive.compactor.initiator.on</name>
    <value>true</value>
</property>

<property>
    <name>hive.compactor.worker.threads</name>
    <value><positive_number></value>
</property>
```

Important

These are the minimum properties required to enable transactions in the standalone Hive metastore. See Hive Transactions on the Apache web site for information about configuring Hive for transactions and additional configuration parameters.

Even though HiveServer2 runs with an embedded metastore, a standalone Hive metastore is required for ACID support to function properly. If you are not using ACID support with HiveServer2, you do not need a standalone metastore.

The default value for `hive.compactor.worker.threads` is `0`. Set this to a positive number to enable Hive transactions. Worker threads spawn MapReduce jobs to perform compactions, but they do not perform the compactions themselves. Increasing the number of worker threads decreases the time that it takes tables or partitions to be compacted. However, increasing the number of worker threads also increases the background load on the Hadoop cluster because they cause more MapReduce jobs to run in the background.

2.5.2. Configuring HiveServer2 for LDAP and for LDAP over SSL

HiveServer2 supports authentication with LDAP and LDAP over SSL (LDAPS). The following topics cover configuration:

- To configure HiveServer2 to use LDAP
- To configure HiveServer2 to use LDAP over SSL (LDAPS)

To configure HiveServer2 to use LDAP:

1. Add the following properties to the `hive-site.xml` file to set the server authentication mode to LDAP:

```
<property>
     <name>hive.server2.authentication</name>
     <value>LDAP</value>
</property>

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

Where LDAP_URL is the access URL for your LDAP server. For example, `ldap://ldap_host_name@xyz.com:389`.

2. Depending on whether or not you use Microsoft Active Directory as your directory service, add the following additional properties to the `hive-site.xml` file:

   - **Other LDAP service types including OpenLDAP**:

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

     Where LDAP_BaseDN is the base LDAP distinguished name for your LDAP server. For example, `ou=dev, dc=xyz, dc=com`.

   - **Active Directory** (AD):

     ```
     <property>
          <name>hive.server2.authentication.ldap.Domain</name>
          <value>AD_Domain</value>
     </property>
     ```

     Where AD_Domain is the domain name of the AD server. For example, `corp.domain.com`.

3. Test the LDAP authentication. For example, use the Beeline client as follows:

   - If the HiveServer2 transport mode is binary (hive.server2.transport.mode=binary), use the following syntax:

     ```
     beeline>!connect
     jdbc:hive2://node1:<port>/default
     ```

     The Beeline client prompts for the user ID and password.

   - If the HiveServer2 transport mode is HTTP (hive.server2.transport.mode=http) and the Thrift path is cliservice (hive.server2.thrift.http.path=cliservice), use the following syntax:

     ```
     beeline>!connect
     jdbc:hive2://node1:<port>/default;transportMode=http;httpPath=cliservice
     ```

To configure HiveServer2 to use LDAP over SSL (LDAPS):

To enable Hive and the Beeline client to use LDAPS, perform the following actions.

Note

Two types of certificates can be used for LDAP over SSL with HiveServer2:

- *CA Certificates*, which are digital certificates that are signed by a Certificate Authority (CA)
- Self-signed certificates

1. Add the following properties to the `hive-site.xml` file to set the server authentication mode to LDAP:

```
<property>
     <name>hive.server2.authentication</name>
     <value>LDAP</value>
</property>

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

   Where LDAP_URL is the access URL for your LDAP server. For example, `ldap://ldap_host_name@xyz.com:389`.

2. Depending on whether or not you use Microsoft Active Directory as your directory service, add the following additional properties to the `hive-site.xml` file:

   - **Other LDAP service types including OpenLDAP**:

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

     Where LDAP_BaseDN is the base LDAP distinguished name for your LDAP server. For example, `ou=dev, dc=xyz, dc=com`.

   - **Active Directory** (AD):

```
<property>
     <name>hive.server2.authentication.ldap.Domain</name>
     <value>AD_Domain</value>
</property>
```

     Where AD_Domain is the domain name of the AD server. For example, `corp.domain.com`.

3. Depending on which type of certificate you are using, perform one of the following actions:

   - **CA certificate**:

     If you are using a certificate that is signed by a CA, the certificate is already included in the default Java trustStore located at `${JAVA_HOME}/jre/lib/security/cacerts` on all of your nodes. If the CA certificate is not present, you must import the certificate to your Java `cacert` trustStore using the following command:

```
keytool -import -trustcacerts -alias <MyHiveLdaps>
-storepass <password> -noprompt -file <myCert>.pem -keystore ${JAVA_HOME}/
jre/lib/security/cacerts
```

If you want to import the CA certificate into another trustStore location, replace `${JAVA_HOME}/jre/lib/security/cacerts` with the `cacert` location that you want to use.

- **Self-signed certificate**:

  If you are using a self-signed digital certificate, you must import it into your Java `cacert` trustStore. For example, if you want to import the certificate to a Java `cacert` location of `/etc/pki/java/cacerts`, use the following command to import your self-signed certificate:

```
keytool -import -trustcacerts -alias <MyHiveLdaps>
-storepass <password> -noprompt -file <myCert>.pem -keystore /etc/pki/
java/cacerts
```

4. If your trustStore is not `${JAVA_HOME}/jre/lib/security/cacerts`, you must set the HADOOP_OPTS environment variable to point to your CA certificate so that the certificate loads when the HDP platform loads.

   Note

   There is no need to modify the `hadoop-env` template if you use the default Java trustStore of `${JAVA_HOME}/jre/lib/security/cacerts`.

   To set this in Ambari:

   a. In the list of services on the left, click **HDFS**.

   b. Select the **Configs** tab.

   c. On the Configs tab page, select the **Advanced** tab.

   d. Scroll down, and expand the **Advanced hadoop-env** section.

   e. Add the following configuration information to the **hadoop-env template** text box:

```
export HADOOP_OPTS="-Djava_net_preferIPv4Stack=true
-Djavax.net.ssl.trustStore=/etc/pki/java/cacerts
-Djavax.net.ssl.trustStorePassword=changeit ${HADOOP_OPTS}"
```

   f. Click **Save**.

5. Restart the HDFS and Hive services.

   To restart these services in Ambari:

   a. Click the service name on the left margin of the page.

   b. On the service page, click **Service Actions**.

   c. Choose **Restart All**.

For more information about restarting components in Ambari, see "Managing Services" in the *Ambari User's Guide*.

6. Test the LDAPS authentication. For example, if you are using the Beeline client, use the following syntax, assuming the HiveServer2 transport mode is binary:

```
beeline>!connect jdbc:hive2://node1:10000/default
```

The Beeline client prompts for the user ID and password.

Note

- Components such as Apache Knox and Apache Ranger do not use the `hadoop-env.sh` template. The configuration files for these components must be set for LDAPS and manually restarted.
- Ambari Hive View does not work with LDAP or LDAPS.

2.6. Securing Apache Hive

Authorization determines whether a user has the required permissions to perform select operations, such as creating, reading, and writing data, as well as editing table metadata. Apache Ranger provides centralized authorization for all HDP components, and Hive also provides three authorization models. Administrators should consider the specific use case when choosing an authorization model.

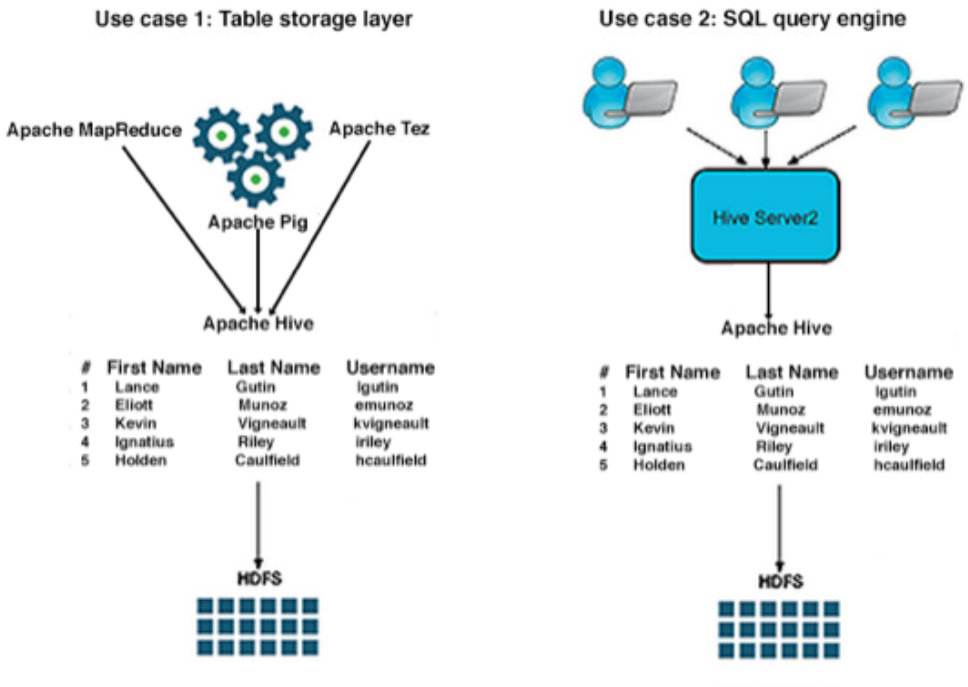
There are two primary use cases for Hive:

- **Table storage layer**

  Many HDP components and underlying technologies, such as Apache Hive, Apache HBase, Apache Pig, Apache MapReduce, and Apache Tez rely on Hive as a table storage layer.

- **SQL query engine**

  Hadoop administrators, business analysts, and data scientists use Hive to run SQL queries, both from the Hive CLI and remotely through a client connecting to Hive through HiveServer2. These users often configure a data analysis tool, such as Tableau, to connect to Hive through HiveServer2.



When using a JDBC or ODBC driver, the value of the hive.server2.enable.doAs configuration property in the `hive.site.xml` file determines the user account that runs a Hive query. The value assigned to this property depends on the desired Hive authorization model and, in the case of storage-based authorization, on the desired use case.

Hive LLAP and the doAs Flag

The architecture of Hive LLAP shares and caches data across many users in much the same way as other MPP or database technologies do. As a result, older file-based security controls do not work with this architecture and doAs is not supported with Hive LLAP. You must use Apache Ranger security policies with a doAs=false setting to achieve secure access via Hive LLAP, while restricting underlying file access so that Hive can access it but unprivileged users cannot.

Apache Ranger and Other Authorization Models

In addition to the centralized authorization provided by Apache Ranger, Hive can use three other authorization models:

Authorization model	Secure?	Fine-grained authorization (column, row level)	Privilege management using GRANT/REVOKE statements	Centralized management GUI
Apache Ranger	Secure	Yes	Yes	Yes
SQL standard-based	Secure	Yes, through privileges on table views	Yes	No

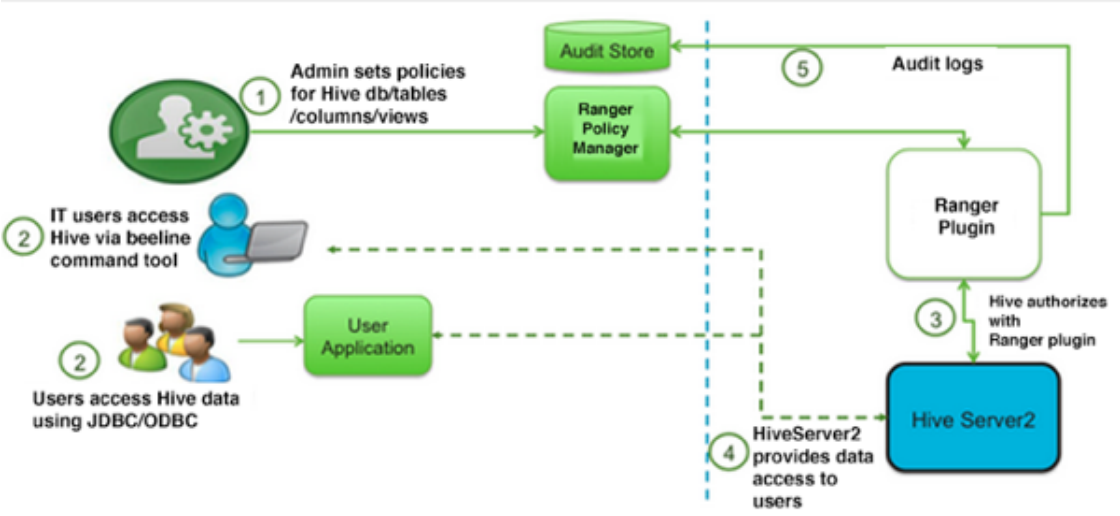
Authorization model	Secure?	Fine-grained authorization (column, row level)	Privilege management using GRANT/REVOKE statements	Centralized management GUI
Storage-based	Secure	No. Authorization at the level of databases, tables, and partitions	No. Table privilege based on HDFS permission	No
Hive default	Not secure. No restriction on which users can run GRANT statements	Yes	Yes	No

Note

Administrators can secure the Hive CLI with Kerberos and by setting permisssions on the HDFS directories where tables reside. The exception to this is storage-based authorization, which does not require managing HDFS permissions and is the most secure authorization model for the Hive CLI.

2.6.1. Authorization Using Apache Ranger Policies

Apache Ranger provides centralized policy management for authorization and auditing of all HDP components, including Hive. All HDP components are installed with a Ranger plugin used to intercept authorization requests for that component, as shown in the following illustration.



Note

Administrators who are responsible for managing access to multiple components are *strongly encouraged* to use the Ranger Policy Manager to configure authorization for Hive rather than using storage-based or SQL standard-based authorization.

Exceptions: There are two primary use cases where administrators might choose to integrate Ranger with SQL standard-based authorization provided by Hive:

- An administrator is responsible for Hive authentication but not authentication for other HDP components
- An administrator wants row-level authentication for one or more table views

In the first use case, an administrator could choose any of the authorization models provided by Hive. The second use case is possible by integrating Ranger with SQL standard-based authorization provided by Hive. Hortonworks recommends that administrators who use both Ranger and SQL standard-based authorization use either whitelisted policies in the Policy Manager or GRANT and REVOKE statements in Hive, but not both. Authentication changes made with GRANT and REVOKE statements appear as updates to the corresponding white policy; there is no need to configure authorization both ways. Ranger also provides an option to disable the use of GRANT and REVOKE statements.

There are two notable differences between Ranger authorization and SQL standard-based authorization:

- Ranger does not have the concept of a role. Instead, Ranger translates roles into users and groups.
- The ADMIN permission in Ranger is the equivalent to the WITH GRANT OPTION in SQL standard-based authorization. However, the ADMIN permission gives the grantee the ability to grant all permissions rather than just the permissions possessed by the grantor. With SQL standard-based authorization, the WITH GRANT OPTION applies only to permissions possessed by the grantor.

For more information about using Ranger to configure Hive authorization, see the Apache Ranger User Guide. For more information about SQL standard-based authorization, see the following sections.

2.6.2. SQL Standard-Based Authorization

SQL standard-based authorization provides fine-grained control using GRANT and REVOKE statements and supports row and column-level access with table views. Granting access to a table view is safer than granting access to the underlying table. This authorization model is disabled for the Hive command line. Secure access from the Hive CLI is not possible because users have direct access to HDFS and can bypass SQL standard-based authorization checks and even disable the authorization model. As the name suggests, this authorization model mimics traditional SQL compliant authorization in relational database systems with the GRANT and REVOKE commands. A user's privileges are checked when she runs a Hive query or command.

For more information about the ongoing work to fully support the SQL-2011 standard, see "SQL Compliance".

Administrators can grant roles as well as privileges. Users can belong to one or more roles. Two roles have special meaning:

- public
- admin

All users belong to the public role. Administrators should use this role in GRANT statements intended to grant a privilege to all users. Administrators should add users who do the work

of a database administrator to the admin role. These users have privileges to run additional commands such as CREATE ROLE and DROP ROLE, and they can access objects without getting explicit access rights. However, users who belong to the admin role need to run the SET ROLE command before using the privileges of the admin role because this role is not included with the current roles by default.

The ownership of a table, view, or database determines who is authorized to perform certain actions. For example, the user who creates a table, view, or database becomes its owner. In the case of tables and views, the owner gets all the privileges with the GRANT option. Administrators can also use the ALTER DATABASE command to specify a role as the owner of a database.

SQL standard-based authorization models consider users with access to the following functionality as privileged:

- Hive CLI
- HDFS commands
- Pig CLI
- **hadoop jar** command
- MapReduce

These tools and commands do not access data through HiveServer2, so SQL standard-based authorization cannot authorize their access. Hortonworks recommends that administrators configure storage-based authorization on the Hive Metastore server to control access to data in Hive tables for these users. The two authorization models are compatible.

Note

Currently, SQL standard-based authorization does not poll groups from LDAP.

2.6.2.1. Configuring SQL Standard-Based Authorization

Prerequisite

You must have permission to run Hive commands as admin.

Steps

Use the following procedure to configure SQL standard-based authorization for Hive:

1. Set the following configuration properties in the `hive-site.xml` file to enable SQL standard-based authorization.
   - `hive.server2.enable.doAs`

     Allows Hive queries to be run by the user who submits the query, rather than by the `hive` user. Must be set to **`false`** for SQL standard-based authorization.
   - `hive.users.in.admin.role`

     Comma-separated list of users assigned to the admin role.

2. Grant the ADMIN privilege to the admin role:

   ```
   GRANT admin TO USER hiveadmin;
   ```

3. Start HiveServer2 with the following command-line option settings:

Command line option	Required value
`hive.security.authorization.manager`	org.apache.hadoop.hive.ql.security.authorization.plugin.sql
`hive.security.authorization.enabled`	true
`hive.security.authenticator.manager`	org.apache.hadoop.hive.ql.security.SessionStateUserAuthenticator
`hive.metastore.uris`	"" (Quotation marks surrounding a single empty space)

   These properties appear in the following snippet of the `hive-site.xml` file:

```
<property>
 <name>hive.security.authorization.manager</name>
 <value>org.apache.hadoop.hive.ql.security.authorization.plugin.sql</ value>
</property>

<property>
 <name>hive.security.authorization.enabled</name>
 <value>true</value>
</property>

<property>
 <name>hive.security.authenticator.manager</name>
 <value>org.apache.hadoop.hive.ql.security.SessionStateUserAuthenticator</
value>
</property>

<property>
 <name>hive.metastore.uris</name>
 <value>""</value>
</property>
```

2.6.3. Required Privileges for Hive Operations

Privileges apply to tables and views, but not databases. The following privileges may be granted and revoked:

- Y = required privilege
- Y + G = required privilege and the ability to grant the privilege to other users

The privileges are required for some Hive operations, as specified in the following table.

Hive Operation	SELECT	INSERT	DELETE	Update	Ownership	Admin	URI privilege (POSIX + ownership)
GRANT						Y	
REVOKE						Y	
SHOW GRANT						Y	
SHOW ROLE GRANT						Y	
CREATE ROLE						Y	

SET ROLE						Y	
DROP ROLE						Y	
CREATE TABLE					Y (of database)		
DROP TABLE					Y		
DESCRIBE TABLE	Y						
SHOW PARTITIONS	Y						
ALTER TABLE LOCATION					Y		Y (for new location)
ALTER PARTITION LOCATION					Y		Y (for new partition location
ALTER TABLE ADD PARTITION		Y					Y (for partition location)
ALTER TABLE DROP PARTITION			Y				
all other ALTER TABLE commands					Y		
TRUNCATE TABLE					Y		
CREATE VIEW	Y + G						
ALTER VIEW PROPERTIES					Y		
ALTER VIEW RENAME					Y		
DROP VIEW PROPERTIES					Y		
DROP VIEW					Y		
ANALYZE TABLE	Y	Y					
SHOW COLUMNS	Y						
SHOW TABLE STATUS	Y						
SHOW TABLE PROPERTIES	Y						
CREATE TABLE AS SELECT	Y (of input)				Y	Y (of database)	
UPDATE TABLE	Y						
CREATE INDEX						Y (of table)	
DROP INDEX						Y	

ALTER INDEX REBUILD						Y	
ALTER INDEX PROPERTIES						Y	
QUERY (INSERT, SELECT queries)	Y (input)	Y (output)	Y (output)				
LOAD		Y (output)	Y (output)				Y (input location)
SHOW CREATE TABLE	Y + G						
CREATE FUNCTION						Y	
DROP FUNCTION						Y	
CREATE MACRO						Y	
DROP MACRO						Y	
MSCK (metastore check)						Y	
ALTER DATABASE						Y	
CREATE DATABASE							Y (for custom location)
EXPLAIN	Y						
DROP DATABASE				Y			

2.6.4. Storage-Based Authorization

As the name implies, storage-based authorization relies on the authorization provided by the storage layer. In the case of an HDP cluster, the storage layer is HDFS, which provides both POSIX and ACL permissions. Hive is one of many HDP components that share storage on HDFS. HCatalog provides all of these components with a single consistent view metadata, and this is why storage-based authorization is enabled in the Hive Metastore server. By enabling this model on the Hive Metastore Server, Hadoop administrators can provide consistent data and metadata authorization. The model controls access to metadata and checks permissions on the corresponding directories of the HDFS file system. Traditional POSIX permissions for the HDFS directories where tables reside determine access to those tables. For example, to alter table metadata for a table stored in HDFS at `/user/hive/warehouse/mytable`, a user must have WRITE permissions on that directory. However, this authorization model doesn't support column-level security.

In addition to the traditional POSIX permissions model, HDFS also provides ACLs, or access control lists, as described in *ACLs on HDFS*. An ACL consists of a set of ACL entries, and each entry names a specific user or group and grants or denies read, write, and execute permissions for the specified user or group. These ACLs are also based on POSIX specifications, and they are compatible with the traditional POSIX permissions model.

HDFS ACL permissions provide administrators with authentication control over databases, tables, and table partitions on the HDFS file system. For example, an administrator can create a role with a set of grants on specific HDFS tables, then grant the role to a group of users. Roles allow administrators to easily reuse permission grants. Hortonworks recommends relying on POSIX permissions and a small number of ACLs to augment the POSIX permissions for exceptions and edge cases.

Note

A file with an ACL incurs additional memory cost to the NameNode due to the alternate algorithm used for permission checks on such files.

2.6.5. Configuring Storage-Based Authorization

Prerequisite

You must have admin role privileges.

Steps

1. Set the following configuration properties in the `hive-site.xml` file to enable storage-based authorization:

Configuration Property	Description
`hive.security.authorization.enabled`	Enables or disables Hive client authorization done as part of query compilation. This property must be set to **`false`** in the `hive-site.xml` file for storage-based authorization, as it is already enabled via checks on metastore API calls.
`hive.server2.enable.doAs`	Allows Hive queries to be run by the user who submits the query rather than the Hive user. Must be set to **`true`** for storage-based access.
`hive.metastore.pre.event.listeners`	Enables Metastore security. Specify the following value: **`org.apache.hadoop.hive.ql.security.authorization.AuthorizationPreEventListener`**.
`hive.security.metastore.authorization.manager`	The class name of the Hive Metastore authorization manager. Specify the following value for storage-based authorization: **`org.apache.hadoop.hive.ql.security.authorization.StorageBasedAuthorizationProvider`**.

These properties appear in the following snippet of the `hive-site.xml` file:

```
<property>
   <name>hive.security.authorization.enabled</name>
   <value>false</value>
</property>

<property>
   <name>hive.security.authorization.manager</name>
   <value>org.apache.hadoop.hive.ql.security.authorization.
StorageBasedAuthorizationProvider</value>
</property>
```

```
<property>
   <name>hive.server2.enable.doAs</name>
   <value>true</value>
</property>

<property>
   <name>hive.metastore.pre.event.listeners</name>
   <name>org.apache.hadoop.hive.ql.security.authorization.
AuthorizationPreEventListener</name>
</property>

<property>
   <name>hive.security.metastore.authorization.manager</name>
   <value>org.apache.hadoop.hive.ql.security.authorization.
StorageBasedAuthorizationProvider</value>
</property>
```

2. Determine the required permissions of the tables and databases in your environment. See the "Permissions for Apache Hive Operations" section for further information.

3. Use either of the following methods to create new tables and databases with appropriate storage-based permissions:

- Create the table or database in the Hive CLI, then manually modify the POSIX permissions using the HDFS file system commands.
- Use the HCatalog CLI

The HCatalog command line tool uses the same syntax as Hive, but creates the table or database with a corresponding directory owned by the user creating it. Assigning a group permission is also supported. However, there are known issues with the HCatalog CLI:

- Some metadata operations do not check for authorization. See Apache JIRA HIVE_3009 for more information.
- Hive performs authorization checks on the client, rather than the server. This allows malicious users to circumvent these checks.
- DDL statements for managing permissions have no effect on storage-based authorization, but they do not return error messages. See Apache JIRA HIVE-3010 for more information.

2.6.6. Permissions for Apache Hive Operations

The following table shows the minimum permissions required for Hive operations using storage-based authorization:

Operation	Database READ Access	Database WRITE Access	Table READ Access	Table WRITE Access
LOAD				X
EXPORT			X	
IMPORT				X
CREATE TABLE		X		

Operation	Database READ Access	Database WRITE Access	Table READ Access	Table WRITE Access
CREATE TABLE AS SELECT		X	X (source table)	
DROP TABLE		X		
SELECT			X	
ALTER TABLE				X
SHOW TABLES	X			

2.6.7. Row-Level Filtering and Column Masking

Row-level filtering and column masking is supported in HDP 2.5 and later versions. This functionality allows you to filter rows from query results based on Apache Ranger policies and the ability to mask data in query results based on Apache Ranger policies.

Row-Level Filtering

To create row-level filtering, a new type of policy has been added to Apache Ranger: Row Level Filter. This filter is very similar to existing access policies and contains filters for specific users, groups, and conditions. The filter must be a valid WHERE clause for the table or view. Each table or view should have its own row-filter policy.

Filters are evaluated in order by priority. You can exclude users, groups, and conditions from row-level filtering.

Note

Wilcard matching of the database or table is not supported.

Column Masking

To create column masking, a new type of policy has been added to Apache Ranger: Masking. This filter supports various types of masking including the following: show last 4 digits, show first 4 digits, hash, show only year, and NULL. You can pick the type of masking for specific users, groups, or conditions. Each column should have its own masking policy.

Masks are evaluated in the order that they are listed in the policy. You can exclude users, groups, or conditions from masking. HDP supports the addition of mask types through configuration and UDFs.

Note

Wildcard matching of the database, table, or column is not supported.

For more information, refer to Row-level Filtering and Column Masking in Hive.

2.7. Troubleshooting

The following is only a small, partial list of issues and recommended solutions.

- **Hive transaction queries on an Oracle database fail with the error `org.apache.hadoop.hive.ql.lockmgr.LockException: No record of lock could be found, may have timed out`**

This error can be caused by a bug in the BoneCP connection pooling library. In this case, Hortonworks recommends that you set the `datanucleus.connectionPoolingType` property to `dbcp` so the DBCP library is used.

- **Error related to character set used for MySQL: "Specified key was too long; max key length is 767 bytes"**

  MySQL is the default database used by the Hive metastore. Depending on several factors, such as the version and configuration of MySQL, Hive developers may encounter an error message similar to the following:

  ```
  An exception was thrown while adding/validating classes) : Specified key was
   too long; max key length is 767 bytes
  ```

  Administrators can resolve this issue by altering the Hive metastore database to use the Latin-1 character set, as shown in the following example:

  ```
  mysql> ALTER DATABASE <metastore_database_name> character set latin1;
  ```

- **Limitations when using the `timestamp.formats` SerDe parameter**

  The `timestamp.formats` SerDe parameter, introduced in HDP 2.3, produces the following behaviors:

  - Displays only 3 decimal digits when it returns values, but it accepts more decimal digits.

    For example, if you run the following commands:

    ```
    drop table if exists src_hbase_ts;

    create table src_hbase_ts( rowkey string, ts1 string, ts2 string, ts3
     string, ts4 string )
    STORED BY 'org.apache.hadoop.hive. hbase. HBaseStorageHandler' WITH
     SERDEPROPERTIES
    ('hbase.columns.mapping' = 'm:ts1,m:ts2,m:ts3,m:ts4') TBLPROPERTIES
     ('hbase.table.name' = 'hbase_ts');

    insert into src_hbase_ts values ('1','2011-01-01T01:01: 01.111111111',
     '2011-01-01T01:01: 01.123456111',
    '2011-01-01T01:01: 01.111111111', '2011-01-01T01:01: 01.134567890');

    drop table if exists hbase_ts_1;

    create external table hbase_ts_1( rowkey string, ts1 timestamp, ts2
     timestamp, ts3 timestamp, ts4 timestamp )
    STORED BY 'org.apache.hadoop.hive. hbase. HBaseStorageHandler' WITH
     SERDEPROPERTIES
    ( 'hbase.columns.mapping' = 'm:ts1,m:ts2,m:ts3,m:ts4', 'timestamp.formats'
     = "yyyy-MM-dd'T'HH:mm:ss.SSSSSSSSS")
    TBLPROPERTIES ('hbase.table.name' = 'hbase_ts');

    select * from hbase_ts_1;
    ```

    The `timestamp.formats` parameter displays:

    ```
    1 2011-01-01 01:01:01.111 2011-01-01 01:01:01.123 2011-01-01 01:01:01.111
     2011-01-01 01:01:01.134
    ```

When the expected output is:

```
1 2011-01-01 01:01:01.111111111 2011-01-01 01:01:01.123456111 2011-01-01
 01:01:01.111111111 2011-0
```

- The `yyyy-MM-dd'T'HH:mm:ss.SSSSSSSSS` format accepts any timestamp data up to `.SSSSSSSSS` decimal digits (9 places to the left of the decimal) instead of only reading data with `.SSSSSSSSS` decimal digits (9 places to the left of the decimal).

  For example, if you run the following commands:

```
drop table if exists src_hbase_ts; create table src_hbase_ts( rowkey
 string, ts1 string, ts2 string, ts3 string, ts4 string )
STORED BY 'org.apache.hadoop. hive. hbase.HBaseStorageHandler' WITH
 SERDEPROPERTIES
('hbase.columns.mapping' = 'm:ts1,m:ts2,m:ts3,m:ts4') TBLPROPERTIES
 ('hbase.table.name' = 'hbase_ts');

insert into src_hbase_ts values ('1','2011-01-01T01:01: 01.111111111',
 '2011-01-01T01:01: 01.111',
'2011-01-01T01:01: 01.11', '2011-01-01T01:01:01.1');

drop table if exists hbase_ts_1;

create external table hbase_ts_1( rowkey string, ts1 timestamp, ts2
 timestamp, ts3 timestamp, ts4 timestamp )
STORED BY 'org.apache.hadoop. hive. hbase.HBaseStorageHandler' WITH
 SERDEPROPERTIES
( 'hbase.columns.mapping' = 'm:ts1,m:ts2,m:ts3,m:ts4', 'timestamp.formats'
 = "yyyy-MM-dd'T'HH:mm:ss.SSSSSSSSS")
TBLPROPERTIES ('hbase.table.name' = 'hbase_ts');

select * from hbase_ts_1;
```

  The actual output is:

```
1 2011-01-01 01:01:01.111 2011-01-01 01:01:01.111 2011-01-01 01:01:01.11
 2011-01-01 01:01:01.1
```

  When the expected output is:

```
1 2011-01-01 01:01:01.111 NULL NULL NULL
```

- **DROP TABLE and DROP PARTITION do not update table content**

  When HDFS is encrypted and the Hadoop trash directory feature is enabled, the DROP TABLE and DROP PARTITION commands might not update the table. In this case, creating a table with the same name as before results in a table with old data.

  When Trash is enabled, the data file for the table should be moved to the Trash bin. If the table is inside an Encryption zone, this move operation is not allowed. For information on HDFS, see HDFS "Data at Rest" Encryption.

  To work around this, use the PURGE command, as shown in the following example.

```
drop table if exists hbase_ts_1 PURGE;
```

2.7.1. JIRAs

Issue tracking for Hive bugs and improvements can be found on the Apache Hive site.

3. Enabling Efficient Execution with Apache Pig and Apache Tez

By default, Apache Pig runs against Apache MapReduce, but administrators and scripters can configure Pig to run against the Apache Tez execution engine to take advantage of more efficient execution and fewer reads of HDFS. Pig supports Tez in all of the following ways:

Command Line	Use the -x command-line option: `pig -x tez`
Pig Properties	Set the following configuration property in the conf/pig.properties file: `exectype=tez`
Java Option	Set the following Java Option for Pig: `PIG_OPTS="-D exectype=tez"`

Users and administrators can use the same methods to configure Pig to run against the default MapReduce execution engine.

Command Line	Use the `-x` command-line option: `pig -x mr`
Pig Properties	Set the following configuration property in the conf/pig.properties file: `exectype=tez`
Java Option	Set the following Java Option for Pig: `PIG_OPTS="-D exectype=tez"`
Pig Script	Use the `set` command: `set exectype=mr;`

There are some limitations to running Pig with the Tez execution engine:

- Queries that include the ORDER BY clause may run slower than if run against the MapReduce execution engine.
- There is currently no user interface that allows users to view the execution plan for Pig jobs running with Tez. To diagnose a failing Pig job, users must read the Application Master and container logs.

Note

Users should configure parallelism before running Pig with Tez. If parallelism is too low, Pig jobs will run slowly. To tune parallelism, add the PARALLEL clause to your PIG statements.

Running a Pig-on-Tez Job with Oozie

To run a Pig job on Tez using Oozie, perform the following configurations:

- Add the following property and value to the `job.properties` file for the Pig-on-Tez Oozie job:

```
<property>
     <name>oozie.action.sharelib.for.pig</name>
     <value>pig, hive</value>
</property>
```

- Create the `$OOZIE_HOME/conf/action-conf/pig` directory and copy the `tez-site.xml` file into it.

4. Accessing Apache Hive with HCatalog and WebHCat

4.1. HCatalog

Hortonworks Data Platform (HDP) deploys HCatalog as a way to access the metadata services of Apache Hive.

HCatalog was designed as a table and storage management service for data created across many different components of Apache Hadoop. However, after release HCatalog in HDP evolved primarily as a way to access Hive services from Apache Pig and MapReduce.

This functionality includes:

- Providing a shared schema and data type mechanism.
- Providing a table abstraction so that users need not be concerned with where or how their data is stored.

Start the HCatalog CLI with the following command:

```
<hadoop-install-dir>\hcatalog-0.5.0\bin\hcat.cmd
```

Note

HCatalog 0.5.0 was the final version released from the Apache Incubator. In March 2013, HCatalog graduated from the Apache Incubator and became part of the Apache Hive project. New releases of Hive include HCatalog, starting with Hive 0.11.0.

For more details about the Apache Hive project, including HCatalog and WebHCat, see the Using Apache Hive chapter and the following resources:

- Hive project home page
- Hive wiki home page
- Hive mailing lists

For information about running *Apache Pig* with HCatalog, see HCatalog LoadStore on the Apache Hive wiki.

To run *MapReduce* jobs on HCatalog-managed tables, see the HCatalog InputOutput page on the wiki.

4.1.1. HCatalog Community Information

For details about HCatalog, see the following resources in the HCatalog documentation set:

- HCatalog Overview

- Installation From Tarball
- HCatalog Configuration Properties
- Load and Store Interfaces
- Input and Output Interfaces
- Reader and Writer Interfaces
- Command Line Interface
- Storage Formats
- Dynamic Partitioning
- Notification
- Storage Based Authorization

4.2. WebHCat

WebHCat is a REST API that supports HTTP-request interfaces, including web GUIs, for users who want an alternative to a command-line environment for working with Hive databases, Pig, and MapReduce.

4.2.1. WebHCat Community Information

Note

WebHCat was originally named Templeton, and both terms may still be used interchangeably. For backward compatibility the Templeton name still appears in URLs and log file names.

For details about WebHCat (Templeton), see the following resources:

- Overview
- Installation
- Configuration
- Reference
  - Resource List
  - GET version
  - GET status
  - GET version
  - DDL Resources: Summary and Commands

- POST mapreduce/streaming
- POST mapreduce/jar
- POST pig
- POST hive
- GET queue/:jobid
- DELETE queue/:jobid

4.2.2. Security for WebHCat

WebHCat currently supports two types of security:

- Default security (without additional authentication)
- Authentication by using Kerberos

Example: HTTP GET :table

The following example demonstrates how to specify the `user.name` parameter in an HTTP GET request:

```
% curl -s 'http://localhost:50111/templeton/v1/ddl/database/default/table/
my_table?user.name=ctdean'
```

Example: HTTP POST :table

The following example demonstrates how to specify the `user.name` parameter in an HTTP POST request

```
% curl -s -d user.name=ctdean \
       -d rename=test_table_2 \
       'http://localhost:50111/templeton/v1/ddl/database/default/table/
          test_table'
```

Security Error

If the `user.name` parameter is not supplied when required, the following security error is returned:

```
{
   "error": "No user found. Missing user.name parameter."
}
```

5. Persistent Read/Write Data Access with Apache HBase

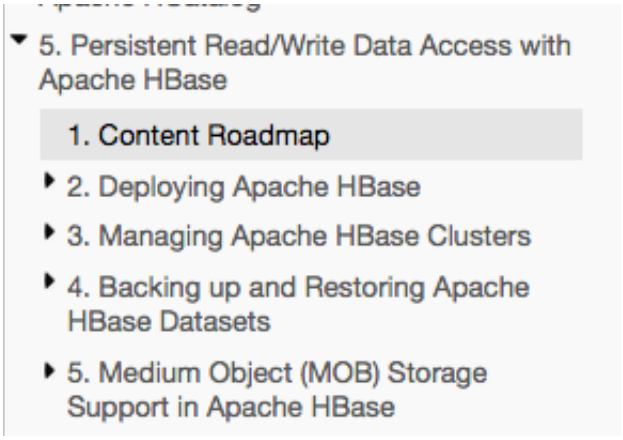
Hortonworks Data Platform (HDP) includes the Apache HBase database, which provides random, persistent access to data in Hadoop. This "NoSQL" database is ideal for scenarios that require real-time analysis and tabular data for end-user applications. Apache HBase can host big data tables because it scales linearly to handle very large (petabyte scale), column-oriented data sets. The data store is predicated on a key-value model that supports low latency reads, writes, and updates in a distributed environment .

As a natively nonrelational database, Apache HBase can combine data sources that use a wide variety of structures and schemas. It is natively integrated with the Apache Hadoop Distributed File System (HDFS) for resilient data storage and is designed for hosting very large tables with sparse data.

In this document:

5.1. Content Roadmap

Expand the nodes on the left-side navigation pane to browse the topics and subtopics covered in this guide. For example, the following screenshot from the website shows that the node for the HBase chapter is expanded and that there are four collapsed nodes under it:



The following table provides links to Apache HBase information resources, as well as some additional Hortonworks Data Platform (HDP) resources that can help you work with and develop for HBase. The table points to resources that are not contained in this *HDP Data Access Guide.*

Table 5.1. HBase Content Roadmap in Other Sources

Type of Information	Resources	Description
Introduction	Welcome to Apache HBase (Source: Apache website)	Provides a short description of Apache HBase and its features. **Important:** Do not use the Download link on the website in place of the Hortonworks distribution of HBase for your HDP Hadoop deployment.
	Apache HBase Architecture Overview and Reference Guide (Source: Apache website)	This link jumps to a summary of how HBase relates to other parts of the Hadoop environment and lists key HBase components. The remainder of the website includes many different explanations, tips, and examples.
	Why We Use HBase series posted on the Apache Software Foundation blog: Scalable Distributed Transactional Queues Medium Data and Universal Data Systems From MySQL to HBase Investing in Big Data: Apache HBase	Contributors from different companies describe why they use HBase, the benefits of the NoSQL database, and case studies. These blog articles include diagrams that illustrate how HBase components work in the Hadoop environment.
Tutorial	Quick Start - Standalone HBase (Source: Apache website)	A getting started tutorial that walks you through the installation and setup of a standalone HBase instance on a local file system. The instance is for instructional purposes only and not intended for production use.
Installation and Upgrade	Installing HDP with the Ambari Web UI (Source: Hortonworks)	Ambari provides an end-to-end management and monitoring solution for your HDP cluster. Using the Ambari Web UI and REST APIs, you can deploy, operate, manage configuration changes, and monitor services for all nodes in your cluster from a central point.
	Installing HDP manually (Source: Hortonworks)	Describes the information and materials you need to get ready to install HDP manually, rather than to install with Ambari.
	Upgrading HDP with Ambari (Source: Hortonworks)	Ambari and the HDP Stack being managed by Ambari can be upgraded independently. This guide provides information on: Getting ready to upgrade Ambari and HDP, Upgrading Ambari, and Upgrading HDP.
	Non-Ambari Cluster Upgrade Guide (Source: Hortonworks)	These instructions cover the upgrade between two minor releases. If you need to upgrade between two maintenance releases, follow the upgrade instructions in the HDP Release Notes.
Performance Tuning	Apache HBase Performance Tuning (Source: Apache website)	Guides you through the many factors that can affect performance of an HBase cluster and how you can adjust your Hadoop stack and other architecture variables to optimize HBase for your goals.
	Tuning G1GC for Your HBase Cluster (Source: Apache Software Foundation)	This is a detailed, step-by-step blog describing how to tune HBase clusters by configuring the Garbage First garbage collector (G1GC).
Monitoring	HBase Service Alerts	Describes the predefined HBase service alerts that are available in Ambari. Review the sections of the

Type of Information	Resources	Description
	(Source: Hortonworks)	chapter preceding "HBase Service Alerts" to learn how to enable and modify alerts.
	Using Grafana (Source: Hortonworks)	Introduces the Grafana-based dashboards of Ambari and how to open the dashboards for each HDP component that is monitored with the visualization tool. The documentation includes a section specifically about HBase dashboards and describes the available metrics about HBase cluster performance.
Security	Hadoop Security Guide (Source: Hortonworks)	Describes how the security features of HDP operate and how to configure settings depending on your Hadoop implementation.
High Availability	High Availability for HBase (Source: Hortonworks)	Guides HBase administrators and developers on how to leverage High Availability capabilities in HBase.
	Enabling HBase High Availability with Ambari (Source: Apache Software Foundation)	Describes how to add an HBase Master component to support High Availability.
Apache Phoenix	Apache Phoenix website (Source: Apache Software Foundation)	Apache Phoenix says "We put the SQL back in NoSQL." This Apache community website provides both an introduction and detailed reference about Phoenix, which is a SQL skin for working with HBase and other Hadoop components.
Hive-HBase Integration	HBaseIntegration wiki (Source: Apache wiki)	Describes how to integrate the two data access components so that HiveQL statements can access HBase tables for both read (SELECT) and write (INSERT) operations.
API Reference	HBase Java API (Source: Hortonworks and Apache)	Reference documentation of HBase Java APIs as generated by Javadoc.

5.2. Deploying Apache HBase

Apache HBase (often simply referred to as *HBase*) operates with many other big data components of the Apache Hadoop environment. Some of these components might or might not be suitable for use with the HBase deployment in your environment. However, two components that must coexist on your HBase cluster are Apache Hadoop Distributed File System (HDFS) and Apache ZooKeeper. These components are bundled with all HDP distributions.

Apache Hadoop Distributed File System (HDFS) is the persistent data store that holds data in a state that allows users and applications to quickly retrieve and write to HBase tables. While technically it is possible to run HBase on a different distributed filesystem, the vast majority of HBase clusters run with HDFS. HDP uses HDFS as its filesystem.

Apache ZooKeeper (or simply *ZooKeeper*) is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services in Hadoop ecosystems. ZooKeeper is essential for maintaining stability for HBase applications in the event of node failures, as well as to to store and mediate updates to important configuration information across the Hadoop cluster.

If you want to use a SQL-like interface to work with the semistructured data of an HBase cluster, a good complement to the other Hadoop components is *Apache Phoenix* (or simply *Phoenix*). Phoenix is a SQL abstraction layer for interacting with HBase. Phoenix enables

you to create and interact with tables in the form of typical DDL and DML statements through its standard JDBC API. HDP supports integration of Phoenix with HBase. See Orchestrating SQL and APIs with Apache Phoenix.

The following table defines some main HBase concepts:

HBase Concept	Description
region	A group of contiguous HBase table rows Tables start with one region, with regions dynamically added as the table grows. Regions can be spread across multiple hosts to provide load balancing and quick recovery from failure. There are two types of regions: primary and secondary. A secondary region is a replicated primary region located on a different RegionServer.
RegionServer	Serves data requests for one or more regions A single region is serviced by only one RegionServer, but a RegionServer may serve multiple regions.
column family	A group of semantically related columns stored together
MemStore	In-memory storage for a RegionServer RegionServers write files to HDFS after the MemStore reaches a configurable maximum value specified with the `hbase.hregion.memstore.flush.size` property in the `hbase-site.xml` configuration file.
Write Ahead Log (WAL)	In-memory log in which operations are recorded before they are stored in the MemStore
compaction storm	A short period when the operations stored in the MemStore are flushed to disk and HBase consolidates and merges many smaller files into fewer large files This consolidation is called *compaction*, and it is usually very fast. However, if many RegionServers reach the data limit specified by the MemStore at the same time, HBase performance might degrade from the large number of simultaneous major compactions. You can avoid this by manually splitting tables over time.

5.2.1. Installation and Setup

You can install and configure HBase for your HDP cluster by using either of the following methods:

- Ambari installation wizard

  The wizard is the part of the Apache Ambari web-based platform that guides HDP installation, including deploying various Hadoop components, such as HBase, depending on the needs of your cluster. See the Ambari Install Guide.

- Manual installation

  You can fetch one of the repositories bundled with HBase and install it on the command line. See the Non-Ambari Installation Guide.

Important

Your HBase installation must be the same version as the one that is packaged with the distribution of the HDP stack version that is deployed across your cluster.

5.2.2. Cluster Capacity and Region Sizing

This section provides information to help you plan the capacity of an HBase cluster and the size of its RegionServers.

5.2.2.1. Node Count and JVM Configuration

The number of nodes in an HBase cluster is typically driven by physical size of the data set and read/write throughput.

5.2.2.1.1. Physical Size of the Data

The physical size of data on disk is affected by the following factors:

Factor Affecting Size of Physical Data	Description
HBase Overhead	The default amount of disk space required for a single HBase table cell. Smaller table cells require less overhead. The minimum cell size is 24 bytes and the default maximum is 10485760 bytes. You can customize the maximum cell size by using the hbase.client.keyvalue.maxsize property in the `hbase-site.xml` configuration file. HBase table cells are aggregated into blocks; you can configure the block size for each column family by using the hbase.mapreduce.hfileoutputformat.blocksize property. The default value is 65536 bytes. You can reduce this value for tables with highly random data access patterns if you want to improve query latency.
Compression	You should choose the data compression tool that is most appropriate to reducing the physical size of your data on disk. Although HBase is not shipped with LZO due to licensing issues, you can install LZO after installing HBase. GZIP provides better compression than LZO but is slower. HBase also supports Snappy.
HDFS Replication	HBase uses HDFS for storage, so replicating HBase data stored in HDFS affects the total physical size of data. A typical replication factor of 3 for all HBase tables in a cluster triples the physical size of the stored data.
RegionServer Write Ahead Log (WAL)	The size of the Write Ahead Log, or WAL, for each RegionServer has minimal impact on the physical size of data: typically fixed at less than half of the memory for the RegionServer. The data size of WAL is usually not configured.

5.2.2.1.2. Read/Write Throughput

The number of nodes in an HBase cluster might also be driven by required throughput for disk reads and writes. The throughput per node greatly depends on table cell size and data request patterns, as well as node and cluster configuration. You can use YCSB tools to test the throughput of a single node or a cluster to determine if read/write throughput should drive the number of nodes in your HBase cluster. A typical throughput for write operations for one RegionServer is 5 through 15 MB/s. Unfortunately, there is no good estimate for read throughput, which varies greatly depending on physical data size, request patterns, and hit rate for the block cache.

5.2.2.2. Region Count and Size

In general, an HBase cluster runs more smoothly with fewer regions. Although administrators cannot directly configure the number of regions for a RegionServer, they can indirectly increase the number of regions in the following ways:

- Increase the size of the MemStore for a RegionServer
- Increase the size of a region

Administrators also can increase the number of regions for a RegionServer by splitting large regions to spread data and the request load across the cluster. HBase enables administrators to configure each HBase table individually, which is useful when tables have different workloads and use cases. Most region settings can be set on a per-table basis by using HTableDescriptor class, as well as by using the HBase CLI. These methods override the properties in the `hbase-site.xml` configuration file. For further information, see Configuring Compactions.

Note

The HDFS replication factor defined in the previous table affects only disk usage and should not be considered when planning the size of regions.

5.2.2.2.1. Increase MemStore size for RegionServer

Use of the RegionServer MemStore largely determines the maximum number of regions for the RegionServer. Each region has one MemStore for each column family, which grows to a configurable size, usually between 128 and 256 MB. Administrators specify this size by using the hbase.hregion.memstore.flush.size property in the `hbase-site.xml` configuration file. The RegionServer dedicates some fraction of total memory to region MemStores based on the value of the hbase.regionserver.global.memstore.size configuration property. If usage exceeds this configurable size, HBase might become unresponsive or compaction storms might occur.

You can use the following formula to estimate the number of regions for a RegionServer:

```
(regionserver_memory_size) * (memstore_fraction) /
((memstore_size) * (num_column_families))
```

For example, assume that your environment uses the following configuration:

- RegionServer with 16 GB RAM (or 16384 MB)
- MemStore fraction of .4
- MemStore with 128 MB RAM
- One column family in table

The formula for this configuration is as follows:

```
(16384 MB * .4) / ((128 MB * 1) = approximately 51 regions
```

The easiest way to decrease the number of regions for this example RegionServer is to increase the RAM of the memstore to 256 MB. The reconfigured RegionServer then has approximately 25 regions, and the HBase cluster runs more smoothly if the reconfiguration is applied to all RegionServers in the cluster. The formula can be used for multiple tables with the same configuration by using the total number of column families in all the tables.

Note

The formula is based on the assumption that all regions are filled at approximately the same rate. If a fraction of the cluster's regions are written to, divide the result by this fraction.

If the data request pattern is dominated by write operations rather than read operations, you should increase the MemStore fraction. However, this increase negatively impacts the block cache.

5.2.2.2.2. Increase Size of Region

The other way to indirectly increase the number of regions for a RegionServer is to increase the size of the region by using the `hbase.hregion.max.filesize` property in the `hbase-site.xml` configuration file. Administrators increase the number of regions for a RegionServer by increasing the specified size at which new regions are dynamically allocated.

Maximum region size is primarily limited by compactions. Very large compactions can degrade cluster performance. The recommended maximum region size is 10 through 20 GB. For HBase clusters running version 0.90.x, the maximum recommended region size is 4 GB and the default is 256 MB. If you are unable to estimate the size of your tables, you should retain the default value. You should increase the region size only if your table cells tend to be 100 KB or larger.

Note

HBase 0.98 introduced stripe compactions as an experimental feature that also enables administrators to increase the size of regions. For more information, see Experimental: Stripe Compactions on the Apache HBase website.

5.2.2.3. Initial Tuning of the Cluster

HBase administrators typically use the following methods to initially configure the cluster:

- Increasing the request handler thread count
- Configuring the size and number of WAL files
- Configuring compactions
- Splitting tables
- Tuning JVM garbage collection in RegionServers

5.2.2.3.1. Increasing the Request Handler Thread Count

Administrators who expect their HBase cluster to experience a high volume request pattern should increase the number of listeners generated by the RegionServers. You can use the `hbase.regionserver.handler.count` property in the `hbase-site.xml` configuration file to set the number higher than the default value of `30`.

5.2.2.3.2. Configuring the Size and Number of WAL Files

HBase uses the Write Ahead Log, or WAL, to recover MemStore data not yet flushed to disk if a RegionServer crashes. Administrators should configure these WAL files to be slightly smaller than the HDFS block size. By default, an HDFS block is 64 MB and a WAL is approximately 60 MB. You should ensure that enough WAL files are allocated to contain the total capacity of the MemStores. Use the following formula to determine the number of WAL files needed:

```
(regionserver_heap_size * memstore fraction) / (default_WAL_size)
```

For example, assume that your environment has the following HBase cluster configuration:

- 16 GB RegionServer heap
- 0.4 MemStore fraction
- 60 MB default WAL size

The formula for this configuration is as follows:

```
(16384 MB * 0.4) / 60 MB = approximately 109 WAL files
```

Use the following properties in the `hbase-site.xml` configuration file to configure the size and number of WAL files:

Configuration Property	Description	Default
hbase.regionserver.maxlogs	Sets the maximum number of WAL files	32
hbase.regionserver.logroll.multiplier	Multiplier of HDFS block size	0.95
hbase.regionserver.hlog.blocksize	Optional override of HDFS block size	Value assigned to actual HDFS block size

Note

If recovery from failure takes longer than expected, try reducing the number of WAL files to improve performance.

5.2.2.3.3. Configuring Compactions

Administrators who expect their HBase clusters to host large amounts of data should consider the effect that compactions have on write throughput. For write-intensive data request patterns, administrators should consider less frequent compactions and more store files per region. Use the hbase.hstore.compaction.min property in the `hbase-site.xml` configuration file to increase the minimum number of files required to trigger a compaction. Administrators opting to increase this value should also increase the value assigned to the hbase.hstore.blockingStoreFiles property because more files will accumulate.

5.2.2.3.4. Splitting Tables

Administrators can split tables during table creation based on the target number of regions per RegionServer to avoid costly dynamic splitting as the table starts to fill. In addition, it ensures that the regions in the pre-split table are distributed across many host machines.

Pre-splitting a table avoids the cost of compactions required to rewrite the data into separate physical files during automatic splitting.

If a table is expected to grow very large, administrators should create at least one region per RegionServer. However, you should not immediately split the table into the total number of desired regions. Rather, choose a low to intermediate value. For multiple tables, you should not create more than one region per RegionServer, especially if you are uncertain how large the table will grow. Creating too many regions for a table that will never exceed 100 MB is not useful; a single region can adequately service a table of this size.

5.2.2.3.5. Tuning JVM Garbage Collection in RegionServers

A RegionServer cannot utilize a very large heap due to the cost of garbage collection. Administrators should specify no more than 24 GB for one RegionServer.

To tune garbage collection in HBase RegionServers for stability, make the following configuration changes:

1. Specify the following configurations in the HBASE_REGIONSERVER_OPTS configuration option in the `/conf/hbase-env.sh` file :

```
-XX:+UseConcMarkSweepGC
-Xmn2500m (depends on MAX HEAP SIZE, but should not be less than 1g and more
 than 4g)
-XX:PermSize=128m
-XX:MaxPermSize=128m
-XX:SurvivorRatio=4
-XX:CMSInitiatingOccupancyFraction=50
-XX:+UseCMSInitiatingOccupancyOnly
-XX:ErrorFile=/var/log/hbase/hs_err_pid%p.log
-XX:+PrintGCDetails
-XX:+PrintGCDateStamps
```

2. Ensure that the block cache size and the MemStore size combined do not significantly exceed `0.5*MAX_HEAP`, which is defined in the HBASE_HEAP_SIZE configuration option of the `/conf/hbase-env.sh` file.

5.2.3. Enabling Multitenancy with Namepaces

A *namespace* is a logical grouping of tables analogous to a database or a schema in a relational database system. With namespaces, a group of users can share access to a set of tables but the users can be assigned different privileges. Similarly, one application can run using the tables in a namespace simultaneously with other applications. Each group of users and each application with access to the instance of the tables defined as a namespace is a *tenant*.

A namespace can support varying ACL-based security modules that can exist among different tenants. Read/write permissions based on groups and users with access to one instance of the namespace function independently from the permissions in another instance.

Unlike relational databases, HBase table names can contain a dot (.) Therefore, HBase uses different syntax, a colon (:), as the separator between the namespace name and table

name. For example, a table with the name `store1` in a namespace that is called `orders` has `store1:orders` as a fully qualified table name. If you do not assign a table to a namespace, then the table belongs to the special `default` namespace.

The namespace file, which contains the objects and data for the tables assigned to a namespace, is stored in a subdirectory of the HBase root directory (`$hbase.rootdir`) on the HDFS layer of your cluster. If `$hbase.rootdir` is at the default location, the path to the namespace file and table is `/apps/hbase/data/data/namespace/table_name`.

Example 5.1. Simple Example of Namespace Usage

A software company develops applications with HBase. Developers and quality-assurance (QA) engineers who are testing the code must have access to the same HBase tables that contain sample data for testing. The HBase tables with sample data are a subset of all HBase tables on the system. Developers and QA engineers have different goals in their interaction with the tables and need to separate their data read/write privileges accordingly.

By assigning the sample-data tables to a namespace, access privileges can be provisioned appropriately so that QA engineers do not overwrite developers' work and vice versa. As tenants of the sample-data table namespace, when developers and QA engineers are logged in as users of this namespace domain they do not access other HBase tables in different domains. This helps ensure that not every user can view all tables on the HBase cluster for the sake of security and ease-of-use.

5.2.3.1. Default HBase Namespace Actions

Tip

If you do not require multitenancy or formalized schemas for HBase data, then do not concern yourself with namespace definitions and assignments. HBase automatically assigns a default namespace when you create a table and do not associate it with a namespace.

The default namespaces are the following:

hbase	A namespace that is used to contain HBase internal system tables
default	A namespace that contains all other tables when you do not assign a specific user-defined namespace

5.2.3.2. Defining and Dropping Namespaces

Important

You can assign a table to only one namespace, and you should ensure that the table correctly belongs to the namespace before you make the association in HBase. You cannot change the namespace that is assigned to the table later.

The HBase shell has a set of straightforward commands for creating and dropping namespaces. You can assign a table to a namespace when you create the table.

create_namespace 'my_ns'	Creates a namespace with the name `my_ns`.
create 'my_ns:my_table', 'fam1'	Creates `my_table` with a column family identified as `fam1` in the `my_ns` namespace.
drop_namespace 'my_ns'	Removes the `my_ns` namespace from the system. The command only functions when there are no tables with data that are assigned to the namespace.

5.2.4. Security Features Available in Technical Preview

The following security features are in Hortonworks Technical Preview:

- *Cell-level access control lists (cell-level ACLs)*: These ACLs are supported in tables of HBase 0.98 and later versions.
- *Column family encryption*: This feature is supported in HBase 0.98 and later versions.

Important

Cell-level ACLs and column family encryption are considered under development. Do not use these features in your production systems. If you have questions about these features, contact Support by logging a case on the Hortonworks Support Portal.

5.3. Managing Apache HBase Clusters

5.3.1. Monitoring Apache HBase Clusters

If you have an Ambari-managed HBase cluster, you can monitor cluster performance with Grafana-based dashboards. The dashboards provide graphical visualizations of data distribution and other boilerplate performance metrics. You can hover over and click graphs to focus on specific metrics or data sets, as well as to redraw visualizations dynamically.

The interactive capabilities of the dashboards can help you to discover potential bottlenecks in your system. For example, you can scan the graphs to get an overview of cluster activity and scroll over a particular time interval to enlarge details about the activity in the time frame to uncover when the data load is unbalanced. Another potential use case is to help you examine if RegionServers need to be reconfigured.

See Using Grafana Dashboards in Ambari for information about how to access the dashboards and for details about what cluster metrics are displayed.

5.3.2. Optimizing Apache HBase I/O

This section introduces HBase I/O and describes several ways to optimize HBase it.

The information in this section is oriented toward basic BlockCache and MemStore tuning. As such, it describes only a subset of cache configuration options. HDP supports additional

BlockCache and MemStore properties, as well as other configurable performance optimizations such as remote procedure calls (RPCs), HFile block size settings, and HFile compaction. For a complete list of configurable properties, see the hbase-default.xml source file in GitHub.

5.3.2.1. An Overview of HBase I/O

The following table describes several concepts related to HBase file operations and memory (RAM) caching.

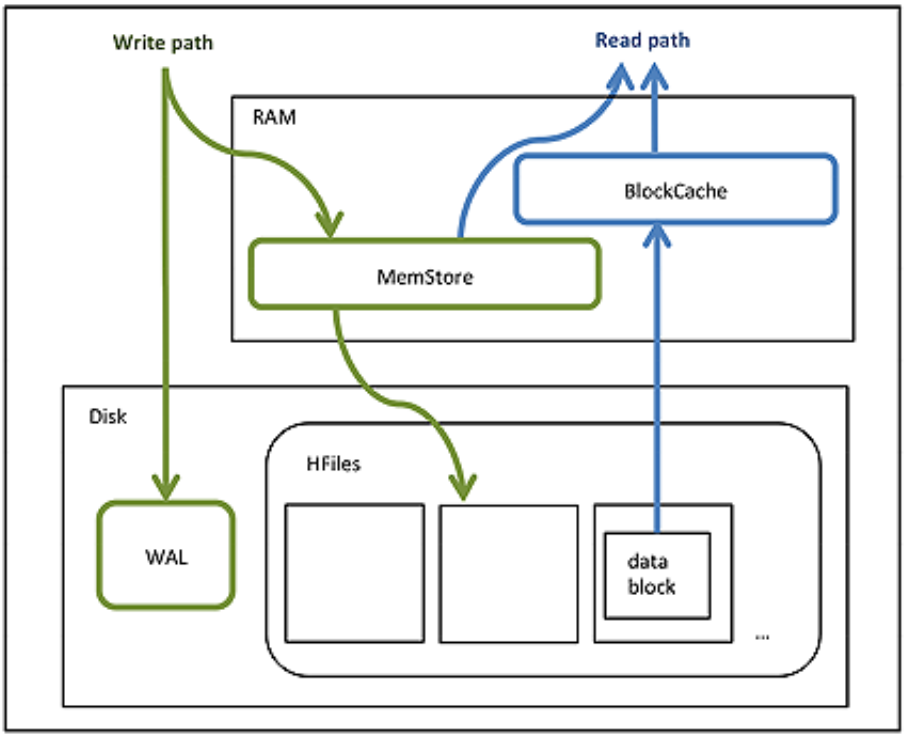
HBase Component	Description
HFile	An HFile contains table data, indexes over that data, and metadata about the data.
Block	An HBase block is the smallest unit of data that can be read from an HFile. Each HFile consists of a series of blocks. (Note: an HBase block is different from an HDFS block or other underlying file system blocks.)
BlockCache	BlockCache is the main HBase mechanism for low-latency random read operations. BlockCache is one of two memory cache structures maintained by HBase. When a block is read from HDFS, it is cached in BlockCache. Frequent access to rows in a block cause the block to be kept in cache, improving read performance.
MemStore	MemStore ("memory store") is in-memory storage for a RegionServer. MemStore is the second of two cache structures maintained by HBase. MemStore improves write performance. It accumulates data until it is full, and then writes ("flushes") the data to a new HFile on disk. MemStore serves two purposes: it increases the total amount of data written to disk in a single operation, and it retains recently written data in memory for subsequent low-latency reads.
Write Ahead Log (WAL)	The WAL is a log file that records all changes to data until the data is successfully written to disk (MemStore is flushed). This protects against data loss in the event of a failure before MemStore contents are written to disk.

BlockCache and MemStore reside in random-access memory (RAM). HFiles and the Write Ahead Log are persisted to HDFS.

The following figure shows these simplified write and read paths:

- During write operations, HBase writes to WAL and MemStore. Data is flushed from MemStore to disk according to size limits and flush interval.
- During read operations, HBase reads the block from BlockCache or MemStore if it is available in those caches. Otherwise, it reads from disk and stores a copy in BlockCache.

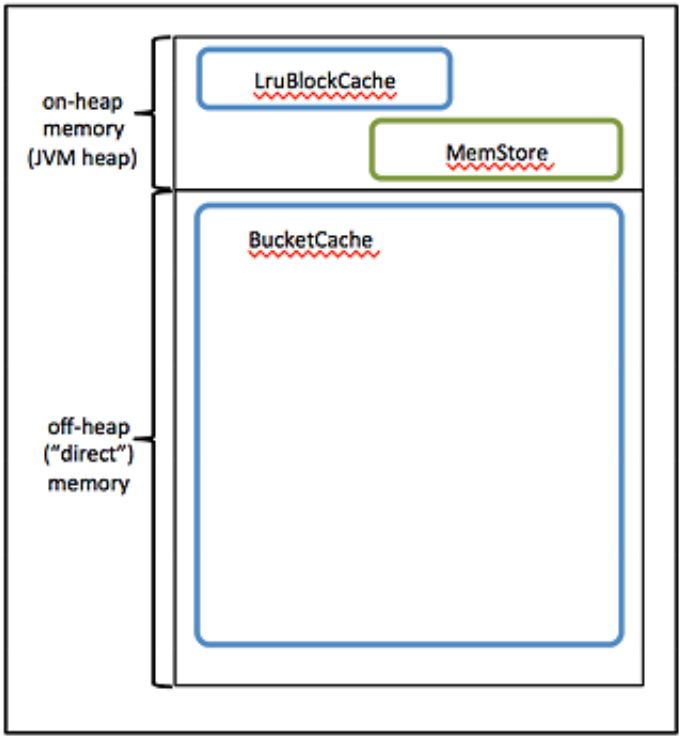
Figure 5.1. HBase Read/Write Operations



By default, BlockCache resides in an area of RAM that is managed by the Java Virtual Machine (JVM) garbage collector; this area of memory is known as *on-heap* memory or the *JVM heap*. The BlockCache implementation that manages the on-heap cache is called *LruBlockCache*.

If you have stringent read latency requirements and you have more than 20 GB of RAM available on your servers for use by HBase RegionServers, consider configuring BlockCache to use both on-heap and *off-heap* memory. *BucketCache* is the off-heap memory equivalent to LruBlockCache in on-heap memory. Read latencies for BucketCache tend to be less erratic than LruBlockCache for large cache loads because BucketCache (not JVM garbage collection) manages block cache allocation. The MemStore always resides in the on-heap memory.

Figure 5.2. Relationship among Different BlockCache Implementations and MemStore



- **Additional notes:**
- BlockCache is enabled by default for all HBase tables.
- BlockCache is beneficial for both random and sequential read operations although it is of primary consideration for random reads.
- All regions hosted by a RegionServer share the same BlockCache.
- You can turn BlockCache caching on or off per column family.

5.3.2.2. Configuring BlockCache

If you have less than 20 GB of RAM available for use by HBase, consider tailoring the default on-heap BlockCache implementation (LruBlockCache) for your cluster.

If you have more than 20 GB of RAM available, consider adding off-heap BlockCache (BucketCache).

To configure either LruBlockCache or BucketCache, start by specifying the maximum amount of on-heap RAM to allocate to the HBase RegionServers on each node. The default is 1 GB, which is too small for production. You can alter the default allocation either with Ambari or in a manual installation:

- *Ambari:* Set a value for the **RegionServer maximum Java heap** size.

- *Manual Installation:* Set the HBASE_HEAPSIZE environment variable in the `hbase-env.sh` file. Specify the value in megabytes. For example, `HBASE_HEAPSIZE=20480` sets the maximum on-heap memory allocation to 20 GB in `hbase-env.sh`. The HBase startup script uses $HBASE_HEAPSIZE to override the default maximum JVM heap size (`-Xmx`).

If you want to configure off-heap BlockCache (BucketCache) only, you are done with configuration.

Additional On-Heap BlockCache (LruBlockCache) Configuration Steps

Determine (or estimate) the proportions of reads and writes in your workload, and use these proportions to specify on-heap memory for BlockCache and MemStore.

The sum of the two allocations must be less than or equal to 0.8. The following table describes the two properties:

Property	Default Value	Description
`hfile.block.cache.size`	0.4	Proportion of maximum JVM heap size (Java `-Xmx` setting) to allocate to BlockCache. A value of 0.4 allocates 40% of the maximum heap size.
`hbase.regionserver.global.memstore.upperLimit`	0.4	Proportion of maximum JVM heap size (Java `-Xmx` setting) to allocate to MemStore. A value of 0.4 allocates 40% of the maximum heap size.

Use the following guidelines to determine the two proportions:

- The default configuration for each property is 0.4, which configures BlockCache for a mixed workload with roughly equal proportions of random reads and writes.
- If the amount of avaiable RAM in the off-heap cache is less than 20 GB, your workload is probably read-heavy. In this case, do not plan to configure off-heap cache, your amount of available RAM is less than 20 GB. In this case, increase the `hfile.block.cache.size` property and decrease the `hbase.regionserver.global.memstore.upperLimit` property so that the values reflect your workload proportions. These adjustments optimize read performance.
- If your workload is write-heavy, decrease the `hfile.block.cache.size` property and increase the `hbase.regionserver.global.memstore.upperLimit` property proportionally.
- As noted earlier, the sum of `hfile.block.cache.size` and `hbase.regionserver.global.memstore.upperLimit` must be less than or equal to 0.8 (80%) of the maximum Java heap size specified by HBASE_HEAPSIZE (`-Xmx`).

  If you allocate more than 0.8 across both caches, the HBase RegionServer process returns an error and does not start.
- Do not set `hfile.block.cache.size` to zero.

  At a minimum, specify a proportion that allocates enough space for HFile index blocks. To review index block sizes, use the RegionServer Web GUI for each server.

Edit the corresponding values in your `hbase-site.xml` files.

Here are the default definitions:

```
<property>
    <name>hfile.block.cache.size</name>
    <value>0.4</value>
    <description>Percentage of maximum heap (-Xmx setting) to allocate to
block
      cache used by HFile/StoreFile. Default of 0.4 allocates 40%.
    </description>
</property>

<property>
    <name>hbase.regionserver.global.memstore.upperLimit</name>
    <value>0.4</value>
    <description>Maximum size of all memstores in a region server before new
      updates are blocked and flushes are forced. Defaults to 40% of heap.
    </description>
</property>
```

If you have less than 20 GB of RAM for use by HBase, you are done with the configuration process. You should restart (or perform a rolling restart on) your cluster and check log files for error messages. If you have more than 20 GB of RAM for use by HBase, consider configuring the variables and properties described next.

5.3.2.2.1. Compressing BlockCache

BlockCache compression caches data and encoded data blocks in their on-disk formats, rather than decompressing and decrypting them before caching. When compression is enabled on a column family, more data can fit into the amount of memory dedicated to BlockCache. Decompression is repeated every time a block is accessed, but the increase in available cache space can have a positive impact on throughput and mean latency.

BlockCache compression is particularly useful when you have more data than RAM allocated to BlockCache, but your compressed data can fit into BlockCache. (The savings must be worth the increased garbage collection overhead and overall CPU load).

If your data can fit into block cache without compression, or if your workload is sensitive to extra CPU or garbage collection overhead, we recommend against enabling BlockCache compression.

Block cache compression is disabled by default.

Important

Before you can use BlockCache compression on an HBase table, compression must be enabled for the table. For more information, see Enable Compression on a ColumnFamily on the Apache website.

To enable BlockCache compression, follow these steps:

1. Set the hbase.block.data.cachecompressed to `true` in the `hbase-site.xml` file on each RegionServer.
2. Restart or perform a rolling restart of your cluster.
3. Check logs for error messages.

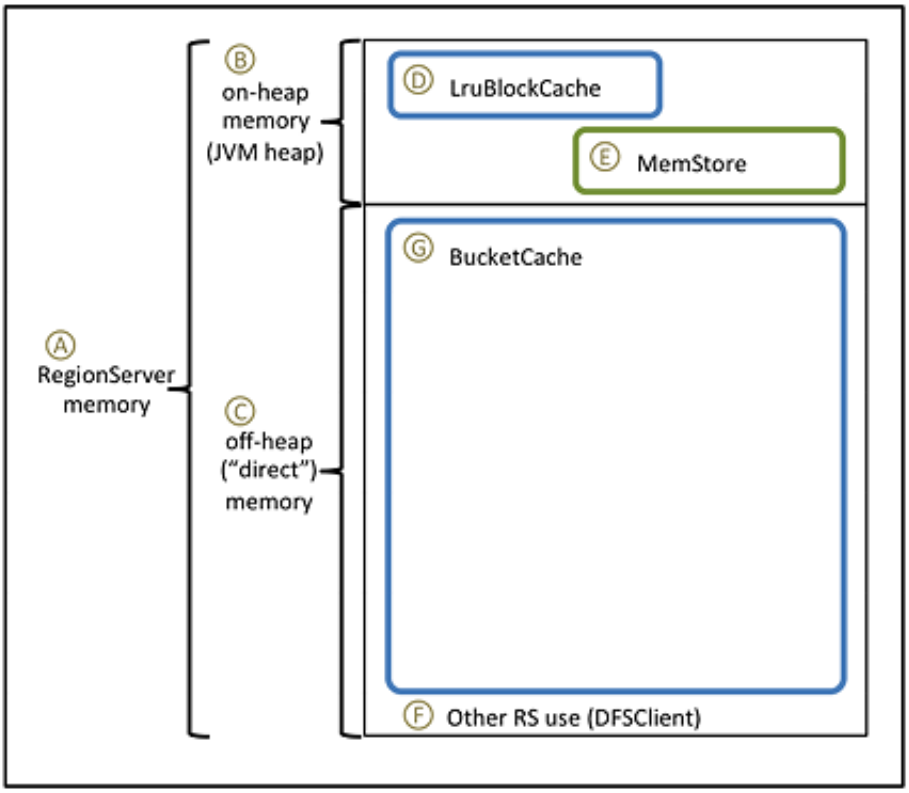
5.3.2.3. Configuring Off-Heap Memory (BucketCache)

Note

Before configuring off-heap memory, complete the tasks in the previous "Configuring BlockCache" section.

To prepare for BucketCache configuration, compare the figure and table below before proceeding to the "Configuring BucketCache" steps.

Figure 5.3. Diagram of Configuring BucketCache



In the following table:

- The first column refers to the elements in the figure.
- The second column describes each element and, if applicable, its associated variable or property name.
- The third column contains values and formulas.
- The fourth column computes values based on the following sample configuration parameters:
  - 128 GB for the RegionServer process (there is additional memory available for other HDP processes)

- A workload of 75% reads, 25% writes
- HBASE_HEAPSIZE = 20 GB (20480 MB)

Note

Most of the following values are specified in megabytes; three are proportions.

Item	Description	Value or Formula	Example
A	Total physical memory for RegionServer operations: on-heap plus off-heap ("direct") memory (MB)	(hardware dependent)	131072
B	The HBASE_HEAPSIZE (-Xmx) property: Maximum size of JVM heap (MB) This value was set when the BlockCache was configured.	Recommendation: 20480	20480
C	The -XX:MaxDirectMemorySize option: Amount of off-heap ("direct") memory to allocate to HBase (MB)	A - B	131072 - 20480 = 110592
Dp	The hfile.block.cache.size property: Proportion of maximum JVM heap size (HBASE_HEAPSIZE, -Xmx) to allocate to BlockCache. The sum of this value plus the hbase.regionserver.global.memstore.size must not exceed 0.8. This value was set when the BlockCache was configured.	(proportion of reads) * 0.8	0.75 * 0.8 = 0.6
Dm	Maximum amount of JVM heap to allocate to BlockCache (MB)	B * Dp	20480 * 0.6 = 12288
Ep	The hbase.regionserver.global.memstore.size property: Proportion of maximum JVM heap size (HBASE_HEAPSIZE, -Xmx) to allocate to MemStore. The sum of this value plus hfile.block.cache.size must be less than or equal to 0.8.	0.8 - Dp	0.8 - 0.6 = 0.2
F	Amount of off-heap memory to reserve for other uses (DFSClient; MB)	Recommendation: 1024 to 2048	2048
G	Amount of off-heap memory to allocate to BucketCache (MB)	C - F	110592 - 2048 = 108544
	The hbase.bucketcache.size	G	108544

	property: Total amount of memory to allocate to the off-heap BucketCache (MB)		
	The `hbase.bucketcache.percentage.in.combinedcache` property: The proportion of memory allocated to off-heap BucketCache, relative to all BlockCache (on- and off-heap)	G / (Dm + G)	108544 / 120832 = 0.89830508474576

5.3.2.3.1. Configuring BucketCache

To configure BucketCache:

1. In the `hbase-env.sh` file for each RegionServer, or in the `hbase-env.sh` file supplied to Ambari, set the `-XX:MaxDirectMemorySize` argument for `HBASE_REGIONSERVER_OPTS` to the amount of direct memory you want to allocate to HBase.

   In the sample configuration, the value would be `110592m` (`-XX:MaxDirectMemorySize` accepts a number followed by a unit indicator; `m` indicates megabytes);

   ```
   HBASE_OPTS="$HBASE_OPTS -XX:MaxDirectMemorySize=110592m"
   ```

2. In the `hbase-site.xml` file, specify BucketCache size and percentage.

   For the sample configuration, the values would be `120832` and `0.89830508474576`, respectively. You can round up the proportion. This allocates space related to the rounding error to the (larger) off-heap memory area.

   ```
   <property>
      <name>hbase.bucketcache.size</name>
      <value>108544</value>
   </property>

   <property>
      <name>hbase.bucketcache.percentage.in.combinedcache</name>
      <value>0.8984</value>
   </property>
   ```

3. In the `hbase-site.xml` file, set `hbase.bucketcache.ioengine` to `offheap` to enable BucketCache:

   ```
   <property>
       <name>hbase.bucketcache.ioengine</name>
       <value>offheap</value>
   </property>
   ```

4. Restart (or perform a rolling restart on) the cluster. It can take a minute or more to allocate BucketCache, depending on how much memory you are allocating. Check logs for error messages.

5.3.3. Importing Data into HBase with Bulk Load

Importing data with a bulk load operation bypasses the HBase API and writes content, properly formatted as HBase data files (HFiles), directly to the file system. Bulk load uses fewer CPU and network resources than using the HBase API for similar work.

Note

The following recommended bulk load procedure uses Apache HCatalog and Apache Pig.

To bulk load data into HBase:

1. Prepare the input file, as shown in the following `data.tsv` example input file:

```
row1 c1 c2
row2 c1 c2
row3 c1 c2
row4 c1 c2
row5 c1 c2
row6 c1 c2
row7 c1 c2
row8 c1 c2
row9 c1 c2
row10 c1 c2
```

2. Make the data available on the cluster, as shown in this continuation of the example:

```
hadoop fs -put data.tsv /tmp/
```

3. Define the HBase schema for the data, shown here as creating a script file called `simple.ddl`, which contains the HBase schema for `data.tsv`:

```
CREATE TABLE simple_hcat_load_table (id STRING, c1 STRING, c2 STRING)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ( 'hbase.columns.mapping' = 'd:c1,d:c2' )
TBLPROPERTIES ( 'hbase.table.name' = 'simple_hcat_load_table'
);
```

4. Create and register the HBase table in HCatalog:

```
hcat -f simple.ddl
```

5. Create the import file.

   The following example instructs Pig to load data from `data.tsv` and store it in `simple_hcat_load_table`. For the purposes of this example, assume that you have saved the following statement in a file named `simple.bulkload.pig`.

```
A = LOAD 'hdfs:///tmp/data.tsv' USING PigStorage('\t') AS (id:chararray,
 c1:chararray,
c2:chararray);
```

```
-- DUMP A;
STORE A INTO 'simple_hcat_load_table' USING org.apache.hive.hcatalog.pig.
HCatStorer();
```

Note

Modify the filenames and table schema for your environment.

6. Execute the following command on your HBase server machine. The command directs Pig to populate the HBase table by using HCatalog `bulkload`.

```
pig -useHCatalog simple.bulkload.pig
```

5.3.4. Using Snapshots

Prior to HBase 0.94.6, the only way to back up or clone a table was to use the CopyTable or ExportTable utility, or to copy all of the HFiles in HDFS after disabling the table. The disadvantage of these methods is that using the first might degrade RegionServer performance, and using the second requires you to disable the table, which means no reads or writes can occur.

HBase snapshot support enables you to take a snapshot of a table without much impact on RegionServers, because snapshot, clone, and restore operations do not involve data copying. In addition, exporting a snapshot to another cluster has no impact on RegionServers.

5.3.4.1. Configuring a Snapshot

Snapshots are enabled by default starting with HBase 0.95, To enable snapshot support in HBase 0.94.6 up to HBase 0.95, set the `hbase.snapshot.enabled` property to `true`. (Snapshots are enabled by default in 0.95+.)

```
<property>
    <name>hbase.snapshot.enabled</name>
    <value>true</value>
</property>
```

5.3.4.2. Taking a Snapshot

As shown in the following example, start the HBase shell and clone the table:

```
$ hbase shell
hbase> snapshot 'myTable', 'myTableSnapshot-122112'
```

5.3.4.3. Listing Snapshots

You can list and describe all snapshots taken as follows:

```
$ hbase shell
hbase> list_snapshots
```

5.3.4.4. Deleting Snapshots

You can remove a snapshot, and the files associated with that snapshot will be removed if they are no longer needed.

```
$ hbase shell
hbase> delete_snapshot 'myTableSnapshot-122112'
```

5.3.4.5. Cloning a Table from a Snapshot

From a snapshot you can create a new table (clone operation) that contains the same data as the original when the snapshot was taken. The clone operation does not involve data copies . A change to the cloned table does not impact the snapshot or the original table.

```
$ hbase shell
hbase> clone_snapshot 'myTableSnapshot-122112', 'myNewTestTable'
```

5.3.4.6. Restoring a Snapshot

The restore operation requires the table to be disabled so that it can be restored to its state when the snapshot was taken, changing both data and schema, if required.

Important

Because replication works at the log level and snapshots work at the file system level, after a restore, the replicas will be in a different state than the master. If you want to use restore, you need to stop replication and redo the bootstrap.

In case of partial data loss due to client issues, you can clone the table from the snapshot and use a MapReduce job to copy the data that you need from the clone to the main one (instead of performing a full restore, which requires the table to be disabled).

The following is an example of commands for a restore operation:

```
$ hbase shell
hbase> disable 'myTable'
hbase> restore_snapshot 'myTableSnapshot-122112'
```

5.3.4.7. Snapshot Operations and ACLs

If you are using security with the AccessController coprocessor, only a global administrator can take, clone, or restore a snapshot. None of these actions capture ACL rights. Restoring a table preserves the ACL rights of the existing table, while cloning a table creates a new table that has no ACL rights until the administrator adds them.

5.3.4.8. Exporting to Another Cluster

The ExportSnapshot tool copies all the data related to a snapshot (HFiles, logs, and snapshot metadata) to another cluster. The tool executes a MapReduce job, similar to **distcp**, to copy files between the two clusters. Because it works at the file system level, the HBase cluster does not have to be online.

The HBase ExportSnapshot tool must be run as user **hbase**. The HBase ExportSnapshot tool uses the temp directory specified by `hbase.tmp.dir` (for example, `/grid/0/var/log/hbase`), created on HDFS with user **hbase** as the owner.

For example, to copy a snapshot called `MySnapshot` to an HBase cluster `srv2` (`hdfs://srv2:8020/hbase`) using 16 mappers, input the following:

```
$ hbase org.apache.hadoop.hbase.snapshot.ExportSnapshot -snapshot MySnapshot -
copy-to
hdfs://yourserver:8020/hbase_root_dir -mappers 16
```

5.4. Backing up and Restoring Apache HBase Datasets

Important

The Apache HBase backup-and-restore feature of HDP is a technical preview and considered under development. Do not use this feature in your production systems. If you have questions regarding this feature, contact Support by logging a case on the Hortonworks Support Portal.

Backup-and-restore is a standard set of operations for many databases. An effective backup-and-restore strategy helps ensure that you can recover data in case of data loss or failures. The HBase backup-and-restore utility helps ensure that enterprises using HBase as a data repository can recover from these types of incidents. Another important feature of the backup-and-restore utility is the ability to restore the database to a particular point-in-time, commonly referred to as a *snapshot*.

The HBase backup-and-restore utility features both *full backups* and *incremental backups*. A full backup is required at least once. The full backup is the foundation on which incremental backups are applied to build iterative snapshots. Incremental backups can be run on a schedule to capture changes over time, for example by using a Cron job. Incremental backup is more cost effective because it only captures the changes. It also enables you to restore the database to any incremental backup version. Furthermore,the utilities also enable table-level data backup-and-recovery if you do not want to restore the entire dataset of the backup.

5.4.1. Planning a Backup-and-Restore Strategy for Your Environment

There are a few strategies you can use to implement backup-and-restore in your environment. The following sections show how they are implemented and identify potential tradeoffs.

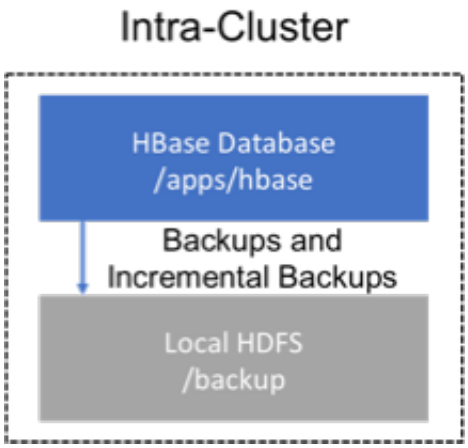
Note

HBase backup-and restore tools are currently not supported on Transparent Data Encryption (TDE)-enabled HDFS clusters. This is related to the Apache HBASE-16178 known issue.

5.4.1.1. Backup within a Cluster

Backup-and-restore within the same cluster is only appropriate for testing. This strategy is not suitable for production unless the underlying HDFS layer is backed up and is reliably recoverable.

Figure 5.4. Intracluster Backup

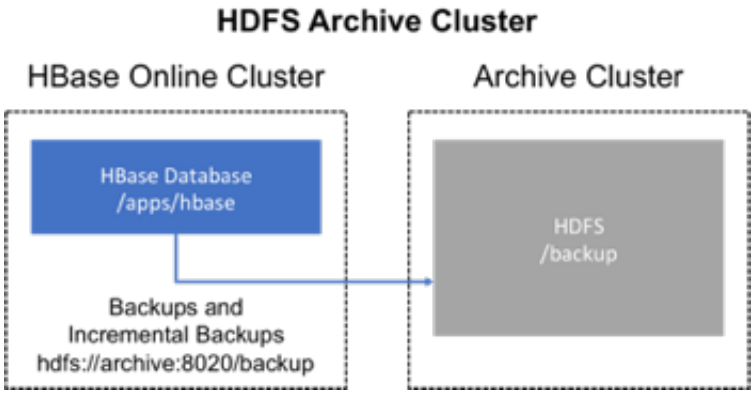


5.4.1.2. Dedicated HDFS Archive Cluster

This strategy provides greater fault tolerance and provides a path towards disaster recovery. In this setting, you will store the backup on a separate HDFS cluster by supplying the backup destination cluster's HDFS URL to the backup utility. You should consider backing up to a different physical location, such as a different data center.

Typically, a backup-dedicated HDFS cluster uses a more economical hardware profile.

Figure 5.5. Backup-Dedicated HDFS Cluster

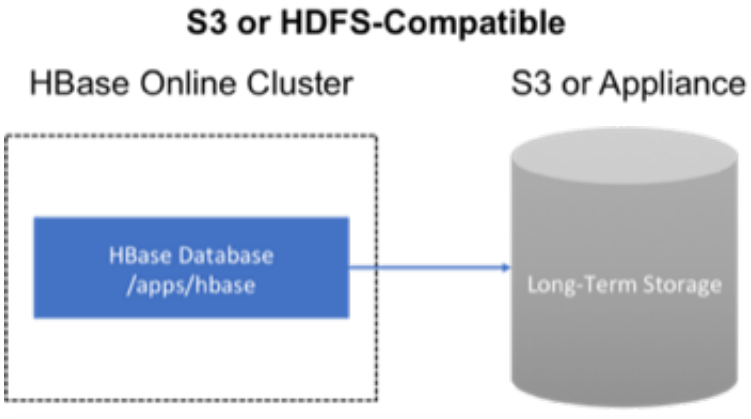


5.4.1.3. Backup to the Cloud or a Storage Vendor

Another approach to safeguarding HBase incremental backups is to store the data on provisioned, secure servers that belong to third-party vendors and that are located off-

site. The vendor can be a public cloud provider or a storage vendor who uses a Hadoop-compatible file system, such as S3 and other HDFS-compatible destinations.

Figure 5.6. Backup to Vendor Storage Solutions



Note

The HBase backup utility does not support backup to multiple destinations. A workaround is to manually create copies of the backed up files from HDFS or S3.

5.4.2. Best Practices for Backup-and-Restore

Formulate a restore strategy and test it. Before you rely on a backup-and-restore strategy for your production environment, identify how backups must be performed, and more importantly, how restores must be performed. Test the plan to ensure that it is workable.

At a minimum, store backup data from a production cluster on a different cluster or server. To further safeguard the data, use a backup location that is at a different site. If you have a unrecoverable loss of data on your primary production cluster as a result of computer system issues, you may be able to restore the data from a different cluster or server at the same site. However, a disaster that destroys the whole site renders locally stored backups useless. Consider storing the backup data and necessary resources (both computing capacity and operator expertise) to restore the data at a site sufficiently remote from the production site. In the case of a catastrophe at the whole primary site (fire, earthquake, etc.), the remote backup site can be very valuable.

Secure a full backup image first. As a baseline, you must complete a full backup of HBase data at least once before you can rely on incremental backups. The full backup should be stored outside of the source cluster. To ensure complete dataset recovery, you must run the restore utility with the option to restore baseline full backup. The full backup is the foundation of your dataset. Incremental backup data is applied on top of the full backup during the restore operation to return you to the point in time when backup was last taken.

Define and use backup sets for groups of tables that are logical subsets of the entire dataset. You can group tables into an object called a *backup set*. A backup set can save time when you have a particular group of tables that you expect to repeatedly back up or restore.

When you create a backup set, you type table names to include in the group. The backup set includes not only groups of related tables, but also retains the HBase backup metadata. Afterwards, you can invoke the backup set name to indicate what tables apply to the command execution instead of entering all the table names individually.

Document the backup-and-restore strategy, and ideally log information about each backup. Document the whole process so that the knowledge base can transfer to new administrators after employee turnover. As an extra safety precaution, also log the calendar date, time, and other relevant details about the data of each backup. This metadata can potentially help locate a particular dataset in case of source cluster failure or primary site disaster. Maintain duplicate copies of all documentation: one copy at the production cluster site and another at the backup location or wherever it can be accessed by an administrator remotely from the production cluster.

5.4.3. Running the Backup-and-Restore Utility

This section details the commands and their arguments of the backup-and-restore utility, as well as example usage based on task.

Important

Prerequisite for Non-Ambari (Manual) Installations of HDP and HBase: You must modify the `container-executor.cfg` configuration file to include the *`allowed.system.users=hbase`* property setting. No spaces are allowed in entries of the `container-executor.cfg` file. Ambari-assisted installations of HDP automatically set the property in the configuration file.

Example of a valid configuration file for backup-and-restore:

```
yarn.nodemanager.log-dirs=/var/log/hadoop/mapred
yarn.nodemanager.linux-container-executor.group=yarn
banned.users=hdfs,yarn,mapred,bin
allowed.system.users=hbase
min.user.id=500
```

Tip

Enter **hbase backup help *command*** in your HBase command-line interface to access the online help that provides basic information about a command and its options.

5.4.3.1. Creating and Maintaining a Complete Backup Image

The first step in running the backup-and-restore utilities is to perform a full backup and to store the data in a separate image from the source. At a minimum, you must do this to get a baseline before you can rely on incremental backups.

Important

For sites using Apache Phoenix: Include the SQL system catalog tables in the backup. In the event that you need to restore the HBase backup, access to the

system catalog tables enable you to resume Phoenix interoperability with the restored data.

Run the following command as **hbase** superuser:

`hbase backup create` {{ full | incremental } {*backup_root_path*} {[*tables*] | [-set *backup_set_name*]}} [[-silent] | [-w *number_of_workers*] | [-b *bandwidth_per_worker*]]

After the command finishes running, the console prints a SUCCESS or FAILURE status message. The SUCCESS message includes a *backup ID*. The backup ID is the Unix time (also known as Epoch time) that the HBase master received the backup request from the client.

Tip

Record the backup ID that appears at the end of a successful backup. In case the source cluster fails and you need to recover the dataset with a restore operation, having the backup ID readily available can save time.

5.4.3.1.1. Required Command-Line Arguments

"full" or "incremental"	Using the `full` argument creates a full backup image. The `incremental` argument directs the command to create an incremental backup that has an image of data changes since the immediately preceding backup, either the full backup or the previous incremental backup.
backup_root_path	The *backup_root_path* argument specifies the full root path of where to store the backup image. Valid prefixes are `hdfs:`, `webhdfs:`, `gpfs:`, and `s3fs:`

5.4.3.1.2. Optional Command-Line Arguments

tables	Table or tables to back up. If no table is specified, all tables are backed up. The values for this argument must be entered directly after the *backup_root_path* argument. Specify tables in a comma-separated list. Namespace wildcards are not supported yet, so to backup a namespace you must enter a full list of tables in the namespace.
-set *backup_set_name*	The `-set` option invokes an existing backup set in the command. See Using Backup Sets for the purpose and usage of backup sets.
-silent	Directs the command to not display progress and completes execution without manual interaction.
-w *number*	Specifies the number of parallel workers to copy data to backup destination (for example, number of map tasks in a MapReduce job).
-b *bandwidth_per_worker*	Specifies the bandwidth of each worker in MB per second.

5.4.3.1.3. Example of Usage

hbase backup create full hdfs://host5:399/data/backup SALES2,SALES3 -w 3

This command creates a full backup image of two tables, SALES2 and SALES3, in the HDFS root path of `//host5:399/data/backup`. The `-w` option specifies that no more than three parallel workers complete the operation.

5.4.3.2. Monitoring Backup Progress

You can monitor a running backup by running the **hbase backup progress** command and specifying the backup ID as an argument.

Run the following command as **hbase** superuser to view the progress of a backup:

```
hbase backup progress {backupId}
```

5.4.3.2.1. Required Command-Line Argument

backupId Specifies the backup that you want to monitor by seeing the progress information. The backup ID argument is case-sensitive.

5.4.3.2.2. Example of Usage

hbase backup progress backupId_1467823988425

This command displays the status of the specified backup.

5.4.3.3. Using Backup Sets

Backup sets can ease the administration of HBase data backups and restores by reducing the amount of repetitive input of table names. You can group tables into a named backup set with the **hbase backup set add** command. You can then use the `-set` option to invoke the name of a backup set in the **hbase backup create** or **hbase backup restore** rather than list individually every table in the group. You can have multiple backup sets.

Note

Note the differentiation between the **hbase backup set add** command and the `-set` option. The **hbase backup set add** command must be run before using the `-set` option in a different command because backup sets must be named and defined before using backup sets as shortcuts.

If you run the **hbase backup set add** command and specify a backup set name that does not yet exist on your system, a new set is created. If you run the command with the name of an existing backup set name, then the tables that you specify are added to the set.

In the command, the backup set name is case-sensitive.

Important

The metadata of backup sets are stored within HBase. If you do not have access to the original HBase cluster with the backup set metadata, then you must specify individual table names to restore the data.

To create a backup set, run the following command as **hbase** superuser:

```
hbase backup set {[add] | [remove] | [list] | [describe] | [delete]}
backup_set_name tables
```

5.4.3.3.1. Subcommands

The following list details subcommands of the **hbase backup set** command.

Note

You must enter one (and no more than one) of the following subcommands after **hbase backup set** to complete an operation. Also, the backup set name is case-sensitive in the command-line utility.

add — Add tables to a backup set. Specify a *backup_set_name* value after this argument to create a backup set.

remove — Removes tables from the set. Specify the tables to remove in the *tables* argument.

list — Lists all backup sets.

describe — Use this subcommand to display on the screen a description of a backup set. The information includes whether the set has full or incremental backups, start and end times of the backups, and a list of the tables in the set. This subcommand must precede a valid value for the *backup_set_name* value.

delete — Deletes a backup set. Enter the value for the *backup_set_name* option directly after the **hbase backup set delete** command.

5.4.3.3.2. Optional Command-Line Arguments

backup_set_name — Use to assign or invoke a backup set name. The backup set name must contain only printable characters and cannot have any spaces.

tables — List of tables (or a single table) to include in the backup set. Enter the table names as a comma-separated list. If no tables are specified, all tables are included in the set.

Tip

Maintain a log or other record of the case-sensitive backup set names and the corresponding tables in each set on a separate or remote cluster, mirroring your backup strategy. This information can help you in case of failure on the primary cluster.

5.4.3.3.3. Example of Usage

hbase backup set add Q1Data TEAM_3,TEAM_4

Depending on the environment, this command results in *one* of the following actions:

- If the `Q1Data` backup set does not exist, a backup set containing tables `TEAM_3` and `TEAM_4` is created.
- If the `Q1Data` backup set exists already, the tables `TEAM_3` and `TEAM_4` are added to the `Q1Data` backup set.

5.4.3.4. Restoring a Backup Image

Run the following command as **hbase** superuser. You can only restore on a live HBase cluster because the data must be redistributed to complete the restore operation successfully.

`hbase restore` {[-set *`backup_set_name`*] | [*`backup_root_path`*] | [backupId] | [*`tables`*]} [[*`table_mapping`*] | [-overwrite] | [-check]]

5.4.3.4.1. Required Command-Line Arguments

-set *`backup_set_name`*	The `-set` option here directs the utility to restore the backup set that you specify in *`backup_set_name`* argument.
`backup_root_path`	The *`backup_root_path`* argument specifies the parent location of the stored backup image.
backupId	The backup ID that uniquely identifies the backup image to be restored.
`tables`	Table or tables to restore. The values for this argument must be entered directly after the `backupId` argument. Specify tables in a comma-separated list.

5.4.3.4.2. Optional Command-Line Arguments

`table_mapping`	Directs the utility to restore data in the tables that are specified in the *`tables`* option. Each table must be mapped prior to running the command. Enter tables as a comma-separated list.
-overwrite	Truncates one or more tables in the target restore location and loads data from the backup image. The existing table must be online before the **hbase restore** command is run to successfully overwrite the data in the table. Compaction is *not* required for the data restore operation when you use the `-overwrite` argument.
-check	Verifies that the restore sequence and dependencies are in working order without actually executing a data restore.

5.4.3.4.3. Example of Usage

hbase restore /tmp/backup_incremental backupId_1467823988425 mytable1,mytable2 -overwrite

This command restores two tables of an incremental backup image. In this example:

- `/tmp/backup_incremental` is the path to the directory containing the backup image.

- `backupId_1467823988425` is the backup ID.
- `mytable1` and `mytable2` are the names of the tables in the backup image to be restored.
- `-overwrite` is an argument that indicates the restored tables overwrite all existing data in the versions of *mytable1* and *mytable2* that exist in the target destination of the restore operation.

5.4.3.5. Administering and Deleting Backup Images

The **hbase backup** command has several subcommands that help with administering backup images as they accumulate. Most production environments require recurring backups, so it is necessary to have utilities to help manage the data of the backup repository. Some subcommands enable you to find information that can help identify backups that are relevant in a search for particular data. You can also delete backup images.

The following list details each **hbase backup *subcommand*** that can help administer backups. Run the full command-subcommand line as **hbase** superuser.

hbase backup history [-n *number_of_backups*]	Displays a log of backup sessions. The information for each session includes backup ID, type (full or incremental), the tables in the backup, status, and start and end time. Specify the number of backup sessions to display with the optional `-n` argument. If no number is specified, the command displays a log of 10 backup sessions.
hbase backup describe {backupId}	Lists the backup image content, time when the backup was taken, whether the backup is full or incremental, all tables in the backup, and backup status. The `backupId` option is required.
hbase backup delete {backupId}	Deletes the specified backup image from the system. The `backup_ID` option is required.

5.4.3.6. Technical Details of Incremental Backup-and-Restore

HBase incremental backups enable more efficient capture of HBase table images than previous attempts at serial backup-and-restore solutions, such as those that only used HBase Export and Import APIs. Incremental backups use Write Ahead Logs (WALs) to capture the data changes since the previous backup was created. A roll log is executed across all RegionServers to track the WALs that need to be in the backup.

After the incremental backup image is created, the source backup files usually are on same node as the data source. A process similar to the DistCp (distributed copy) tool is used to move the source backup files to the target filesystems. When a table restore operation starts, a two-step process is initiated. First, the full backup is restored from the full backup image. Second, all WAL files from incremental backups between the last full backup and the incremental backup being restored are converted to HFiles, which the HBase Bulk Load utility automatically imports as restored data in the table.

You can only restore on a live HBase cluster because the data must be redistributed to complete the restore operation successfully.

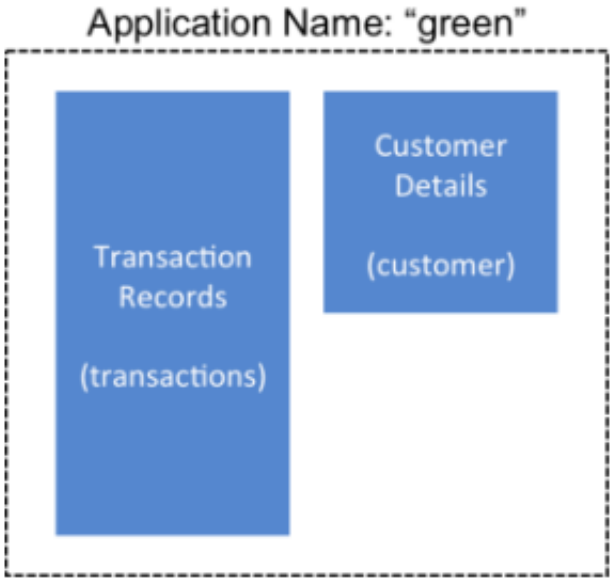
5.4.3.7. Scenario: Safeguarding Application Datasets on Amazon S3

This scenario describes how a hypothetical retail business uses backups to safeguard application data and then restore the dataset after failure.

The HBase administration team uses backup sets to store data from a group of tables that have interrelated information for an application called *green*. In this example, one table contains transaction records and the other contains customer details. The two tables need to be backed up and be recoverable as a group.

The admin team also wants to ensure daily backups occur automatically.

Figure 5.7. Tables Composing the Backup Set



The following is an outline of the steps and examples of commands that are used to backup the data for the *green* application and to recover the data later. All commands are run when logged in as **hbase** superuser.

1. A backup set called *green_set* is created as an alias for both the *transactions* table and the *customer* table. The backup set can be used for all operations to avoid typing each table name. The backup set name is case-sensitive and should be formed with only printable characters and without spaces.

```
$ hbase backup set add green_set transactions
$ hbase backup set add green_set customer
```

2. The first backup of *green_set* data must be a full backup. The following command example shows how credentials are passed to Amazon S3 and specifies the file system with the **s3a:** prefix.

```
$ ACCESS_KEY=ABCDEFGHIJKLMNOPQRST
$ SECRET_KEY=0123456789abcdefghijklmnopqrstuvwxyzABCD
$ sudo -u hbase hbase backup create full \
  s3a://$ACCESS_KEY:$SECRET_KEY@prodhbasebackups/backups -set green_set
```

3. Incremental backups should be run according to a schedule that ensures essential data recovery in the event of a catastrophe. At this retail company, the HBase admin team decides that automated daily backups secures the data sufficiently. The team decides that they can implement this by modifying an existing Cron job that is defined in `/etc/crontab`. Consequently, IT modifies the Cron job by adding the following line:

```
@daily hbase /path/to/hbase/bin/hbase backup create incremental
s3a://$ACCESS_KEY:$SECRET_KEY@prodhbasebackups/backups -set green_set
```

4. A catastrophic IT incident disables the production cluster that the *green* application uses. An HBase system administrator of the backup cluster must restore the *green_set* dataset to the point in time closest to the recovery objective.

   Note

   If the administrator of the backup HBase cluster has the backup ID with relevant details in accessible records, the following search with the **hadoop fs -ls** command and manually scanning the backup ID list can be bypassed. Consider continuously maintaining and protecting a detailed log of backup IDs outside the production cluster in your environment.

   The HBase administrator runs the following command on the directory where backups are stored to print a list of successful backup IDs on the console:

```
hadoop fs -ls -t /prodhbasebackups/backups
```

5. The admin scans the list to see which backup was created at a date and time closest to the recovery objective. To do this, the admin converts the calendar timestamp of the recovery point in time to Unix time because backup IDs are uniquely identified with Unix time. The backup IDs are listed in reverse chronological order, meaning the most recent successful backup appears first.

   The admin notices that the following line in the command output corresponds with the *green_set* backup that needs to be restored:

```
/prodhbasebackups/backups/backupId_1467823988425
```

6. The admin restores *green_set* invoking the backup ID and the `-overwrite` option. The `-overwrite` option truncates all existing data in the destination and populates the tables with data from the backup dataset. Without this flag, the backup data is appended to the existing data in the destination. In this case, the admin decides to overwrite the data because it is corrupted.

```
$ sudo -u hbase hbase restore -set green_set \
  s3a://$ACCESS_KEY:$SECRET_KEY@prodhbasebackups/backups
 backupId_1467823988425 \ -overwrite
```

5.5. Medium Object (MOB) Storage Support in Apache HBase

An HBase table becomes less efficient once any cell in the table exceeds 100 KB of data. Objects exceeding 100 KB are common when you store images and large documents, such as email attachments, in HBase tables. But you can configure Hortonworks Data Platform (HDP) HBase to support tables with cells that have medium-size objects, also known as *medium objects* or more commonly as *MOBs*, to minimize the performance impact that objects over 100 KB can cause. MOB support operates by storing a reference of the object data within the main table. The reference in the table points toward external HFiles that contain the actual data, which can be on disk or in HDFS.

To enable MOB storage support for a table column family, you can choose one of two methods. One way is to run the table **create** command or the table **alter** command with MOB options in the HBase shell. Alternatively, you can set MOB parameters in a Java API.

5.5.1. Enabling MOB Storage Support

You can enable MOB storage support and configure the MOB threshold by using one of two different methods. If you do not specify a MOB size threshold, the default value of 100 KB is used.

Tip

While HBase enforces no maximum-size limit for a MOB column, generally the best practice for optimal performance is to limit the data size of each cell to 10 MB.

Prerequisites:

- **hbase** superuser privileges
- HFile version 3, which is the default format of HBase 0.98+.

Method 1: Configure options in the command line

Run the table **create** command or the table **alter** command and do the following:

- Set the `IS_MOB` option to **`true`**.
- Set the `MOB_THRESHOLD` option to the number of bytes for the threshold size above which an object is treated as a medium-size object.

Following are a couple of HBase shell command examples:

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

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

Method 2: Invoke MOB support parameters in a Java API

You can use the following parameters in a Java API to enable and configure MOB storage support. The second parameter (*hcd.setMobThreshold*) is optional.

If you invoke the MOB threshold parameter, substitute *bytes* with the value for the number of bytes for the threshold size at which an object is treated as a medium-size object. If you omit the parameter when you enable MOB storage, the threshold value defaults to 102400 (100 KB).

- *hcd.setMobEnabled(true);*
- *hcd.setMobThreshold(bytes);*

Following is a Java API example:

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

5.5.2. Testing the MOB Storage Support Configuration

Run the **org.apache.hadoop.hbase.IntegrationTestIngestWithMOB** utility to test the MOB storage configuration. Values in the command options are expressed in bytes.

Following is an example that uses default values (in bytes):

```
$ sudo -u hbase hbase org.apache.hadoop.hbase.IntegrationTestIngestWithMOB \
-threshold 1024 \
-minMobDataSize 512 \
-maxMobDataSize threshold * 5 \
```

5.5.3. Tuning MOB Storage Cache Properties

Opening a MOB file places corresponding HFile-formatted data in active memory. Too many open MOB files can cause a RegionServer to exceed the memory capacity and cause performance degradation. To minimize the possibility of this issue arising on a RegionServer, you might need to tune the *MOB file reader cache* to an appropriate size so that HBase scales appropriately.

The MOB file reader cache is a least recently used (LRU) cache that keeps only the most recently used MOB files open. Refer to the MOB Cache Properties table for variables that can be tuned in the cache. MOB file reader cache configuration is specific to each RegionServer, so assess and change, if needed, each RegionServer individually. You can use either one of the two following methods.

Method 1: Enter property settings using Ambari

1. In Ambari select **Advanced** tab > **Custom HBase-Site** > **Add Property**.
2. Enter a MOB cache property in the **Type** field.
3. Complete the **Value** field with your configuration setting.

Method 2: Enter property settings directly in the `hbase-site.xml` file

1. Open the RegionServer's `hbase-site.xml` file. The file is usually located under `/etc/hbase/conf` .
2. Add the MOB cache properties to the RegionServer's `hbase-site.xml` file.
3. Adjust the parameters or use the default settings.
4. Initiate a restart or rolling restart of the RegionServer. For more information about rolling restarts, see the Rolling Restart section of the online *Apache HBase Reference Guide*.

Table 5.2. MOB Cache Properties

Property and Default Value	Description
hbase.mob.file.cache.size Default Value: 1000	Number of opened file handlers to cache. A larger value enhances read operations by providing more file handlers per MOB file cache and reduce frequent file opening and closing. However, if the value is set too high, a "too many opened file handers" condition can occur.
hbase.mob.cache.evict.period Default Value: 3600	The amount of time (in seconds) after which an unused file is evicted from the MOB cache.
hbase.mob.cache.evict.remain.ratio Default Value: 0.5f	A multiplier (between 0.0 and 1.0) that determines how many files remain cached after the hbase.mob.file.cache.size property threshold is reached. The default value is 0.5f, which indicates that half the files (the least-recently used ones) are evicted.

5.6. HBase Quota Management

Important

The HBase storage quota feature of HDP is a **technical preview** and considered under development. Do not use this feature in your production systems. If you have questions regarding this feature, contact Support by logging a case on the Hortonworks Support Portal.

In a multitenant HBase environment, ensuring that each tenant can use only its allotted portion of the system is key in meeting SLAs. Two types of HBase quotas are well established: throttle quota and number-of tables-quota. These two quotas can regulate users and tables.

As of version 2.6, HDP has an additional quota type: a filesystem space quota. You can use file-system quotas to regulate the usage of filesystem space on namespaces or at the table level.

Table 5.3. Quota Support Matrix

Quota Type	Resource Type	Purpose	Namespace applicable?	Table applicable?	User applicable?
Throttle	Network	Limit overall network throughput and number of RPC requests	Yes	Yes	Yes

Quota Type	Resource Type	Purpose	Namespace applicable?	Table applicable?	User applicable?
New space	Storage	Limit amount of storage used for table or namespaces	Yes	Yes	No
Number of tables	Metadata	Limit number of tables for each namespace or user	Yes	No	Yes
Numbr of regions	Metadata	Limit number of regions for each namespace	Yes	No	No

5.6.1. Setting Up Quotas

Prerequisite

`hbase` superuser privileges

About this Task

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

Steps

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

```
$hbase_shell> set_quota TYPE => {quota_type,arguments}
```

General Quota Syntax

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

Timeframes — Can be expressed in the following units of time:

- **`sec`** (second)
- **`min`** (minute)
- **`hour`**
- **`day`**

Request sizes and space limit — Can be expressed in the following units:

- **`B`**: bytes
- **`K`**: kilobytes
- **`M`**: megabytes
- **`G`**: gigabytes
- **`P`**: petabytes

	When no size units is included, the default value is bytes.
Number of requests	Expressed as integer followed by the string request
Time limits	Expressed as requests per unit-of-time or size per unit-of-time *Examples*: **`10req/day`** or **`100P/hour`**
Number of tables or regions	Expressed as integers

5.6.2. Throttle Quotas

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

Examples of Adding Throttle Quotas Commands

Limit user u1 to 10 requests per second globally:

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

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

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

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

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

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

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

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

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

Limit request to namespaces:

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

Limit request to tables:

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

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

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

Examples of Listing Throttle Quotas Commands

Show all quotas:

```
hbase> list_quotas
```

Show all quotas applied to user bob:

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

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

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

Show all quotas and filter by table or namespace:

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

Examples of Updating and Deleting Throttle Quotas Commands

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

```
hbase> set_quota TYPE => THROTTLE, USER => 'u1', LIMIT => NONE
```

```
hbase> set_quota TYPE => THROTTLE, USER => 'u1', NAMESPACE => 'ns2', LIMIT =>
 NONE
```

```
hbase> set_quota TYPE => THROTTLE, THROTTLE_TYPE => WRITE, USER => 'u1', LIMIT
 => NONE
```

```
hbase> set_quota USER => 'u1', GLOBAL_BYPASS => true
```

5.6.3. Space Quotas

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

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

NO_INSERTS	Prohibits new data from being ingested (for example, data from **Put**, **Increment**, and **Append** operations are not ingested).
NO_WRITES	Performs the same function as **`NO_INSERTS`** but **Delete** operations are also prohibited.
NO_WRITES_COMPACTIONS	Performs the same function as **`NO_INSERTS`** but compactions are also prohibited.

DISABLE Diables tables.

Examples of Adding Space Quotas

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

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

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

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

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

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

Listing Space Quotas

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

Examples of Updating and Deleting Space Quotas

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

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

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

5.6.4. Quota Enforcement

When a quota limit is exceeded, the Master server instructs RegionServers to enable an enforcement policy for the namespace or table that violated the quota. It is important to note the storage quota is not reported in real-time. There is a window when threshold is reached on RegionServers but the threshold accounted for on the Master server is not updated.

Important

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

5.6.5. Quota Violation Policies

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

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

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

The following describes quota violation policies that you might consider.

Disabling Tables

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

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

Rejecting All WRITE Operations, Bulk Imports, and Compactions

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

Rejecting All WRITE Operations and Bulk Imports

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

Allowing DELETE Operations But Rejecting WRITE Operations and Bulk Imports

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

5.6.6. Impact of Quota Violation Policy

"Live" Write Access

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

"Bulk" Write Access

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

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

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

Read Access

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

Metrics and Insight

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

Overlapping Quota Policies

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

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

Table 5.4. Scenario 1: Overlapping Quota Policies

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

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

Table 5.5. Scenario 2: Overlapping Quota Policies

Object	Quota	Storage Utilization
Namespace *n1*	100 GB	60 GB
Table *n1.t1*	10 GB	15 GB

Object	Quota	Storage Utilization
Table *n1.t2*	(not set)	30 GB
Table *n1.t3*	(not set)	15 GB

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

Table 5.6. Scenario 3: Overlapping Quota Policies

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

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

Table 5.7. Scenario 4: Overlapping Quota Policies

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

5.6.7. Number-of-Tables Quotas

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

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

Create namespace *ns1* with a maximum of 5 tables

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

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

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

Show quota information for namespace *ns1*

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

Alter existing namespace *ns1* to remove a quota

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

5.6.8. Number-of-Regions Quotas

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

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

Create namespace *ns1* with a maximum of 5 tables

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

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

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

Show quota information for namespace *ns1*

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

Alter existing namespace *ns1* to remove a quota

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

5.7. HBase Best Practices

You should limit the value of each HBase RowKey length to 32767 bytes. Exceeding this value causes an exceptional error while indexing the table. This restriction is applicable in addition to the maximum allowed size of an individual cell, inclusive of value and all key components as defined by the hbase.server.keyvalue.maxsize value. The default value for RowKey length is 1 MB, which avoids server memory errors.

6. Orchestrating SQL and APIs with Apache Phoenix

Apache Phoenix is a SQL abstraction layer for interacting with Apache HBase and other Hadoop components. Phoenix lets you create and interact with tables in the form of typical DDL/DML statements via its standard JDBC API. With the driver APIs, Phoenix translates SQL to native HBase API calls. Consequently, Phoenix provides a SQL skin for working with data and objects stored in the NoSQL schema of HBase.

This Phoenix documentation focuses on interoperability with HBase. For more information about Phoenix capabilities, see the Apache Phoenix website.

6.1. Enabling Phoenix and Interdependent Components

If you have a Hortonworks Data Platform installation with Ambari, then no separate installation is required for Phoenix.

To enable Phoenix with Ambari:

1. Open Ambari.
2. Select *Services* tab > *HBase* > *Configs* tab.
3. Scroll down to the Phoenix SQL settings.
4. (Optional) Reset the Phoenix Query Timeout.
5. Click the *Enable Phoenix* slider button.

If you installed Hortonworks Data Platform manually and did not include the Phoenix component, see Installing Apache Phoenix.

Important

Your Phoenix installation must be the same version as the one that is packaged with the distribution of the HDP stack version that is deployed across your cluster.

6.2. Thin Client Connectivity with Phoenix Query Server

The Phoenix Query Server (PQS) is a component of the Apache Phoenix distribution. PQS provides an alternative means to connect directly. PQS is a stand-alone server that converts custom API calls from "thin clients" to HTTP requests that make use of Phoenix capabilities. This topology offloads most computation to PQS and requires a smaller client-

side footprint. The PQS client protocol is based on the Avatica component of Apache Calcite.

6.2.1. Securing Authentication on the Phoenix Query Server

You can enable Kerberos-based authentication on PQS with Ambari. If you chose to install HDP manually instead, see Configuring Phoenix Query Server to enable the Kerberos protocol.

6.3. Selecting and Obtaining a Client Driver

You have two options to develop an application that works with Phoenix, depending on the client-server architecture:

Without Phoenix Query Server: If your environment does not have a PQS layer, applications that connnect to Phoenix must use the Phoenix JDBC client driver.

With Phoenix Query Server: PQS is an abstraction layer that enables other languages such as Python and GoLang to work with Phoenix. The layer provides a protocol buffer as an HTTP wrapper around Phoenix JDBC. You might prefer to use a non-Java client driver for one of various reasons, such as to avoid the JVM footprint on the client or to develop with a different application framework.

To obtain the appropriate driver for application development:

JDBC Driver	Use the `/usr/hdp/current/phoenix-client/phoenix-client.jar` file in the Hortonworks Phoenix server-client repository . If you use the repository, download the JAR file corresponding to your installed HDP version. With Ambari, you can determine the HDP version by using the Versions tab. Alternatively, run the **hadoop version** command to print information displaying the HDP version.
JDBC Driver as a Maven dependency	See Download the HDP Maven Artifacts for Maven artifact repositories that are available for HDP.
Microsoft .NET Driver	Download and install a NuGet package for the Microsoft .NET Driver for Apache Phoenix and Phoenix Query Server. *Note:* Operability with this driver is a Hortonworks Technical Preview and considered under development. Do not use this feature in your production systems. If you have questions regarding this feature, contact Support by logging a case on the Hortonworks Support Portal.
Other non-Java drivers	Other non-JDBC Drivers for Phoenix are available as HDP add-ons and on other websites, but they are not currently supported by Hortonworks. You can find compatible client drivers by constructing a web search string consisting of "avatica" and the name of an application programming

language that you want to use. Example: **`avatica python`** .

6.4. Creating and Using User-Defined Functions (UDFs) in Phoenix

With a user-defined function (UDF), you can extend the functionality of your SQL statements by creating scalar functions that operate on a specific tenant. For details about creating, dropping, and how to use UDFs for Phoenix, see User-defined functions on the Apache website.

6.5. Mapping Phoenix Schemas to HBase Namespaces

You can map a Phoenix schema to an HBase namespace to gain multitenancy features in Phoenix.

HBase, which is often the underlying storage engine for Phoenix, has namespaces to support multitenancy features. Multitenancy helps an HBase user or administrator perform access control and quota management tasks. Also, namespaces enable tighter control of where a particular data set is stored on RegionsServers. See Enabling Multitenancy with Namepaces for further information.

Prior to HDP 2.5, Phoenix tables could not be associated with a namespace other than the `default` namespace.

6.5.1. Enabling Namespace Mapping

Important

After you set the properties to enable the mapping of Phoenix schemas to HBase namespaces, reverting the property settings renders the Phoenix database unusable. Test or carefully plan the Phoenix to HBase namespace mappings before implementing them.

To enable Phoenix schema mapping to a non-default HBase namespace:

1. Set the phoenix.schema.isNamespaceMappingEnabled property to `true` in the `hbase-site.xml` file of both the client and the server.

2. Restart the HBase Master and RegionServer processes.

Note

You might not want to map Phoenix system tables to namespaces because there are compatibility issues with your current applications. In this case, set

the phoenix.schema.mapSystemTablesToNamespace property of the `hbase-site.xml` file to `false`.

Namespace Mapping Properties in the `hbase-site.xml` File

phoenix.schema.isNamespaceMappingEnabled
: Enables mapping of tables of a Phoenix schema to a non-default HBase namespace. To enable mapping of schema to a non-default namespace, set the value of this property to `true`. Default setting for this property is `false`.

phoenix.schema.mapSystemTablesToNamespace
: With `true` setting (default): After namespace mapping is enabled with the other property, all system tables, if any, are migrated to a namespace called `system`.

  With `false` setting: System tables are associated with the `default` namespace.

6.5.2. Creating New Schemas and Tables with Namespace Mapping

You can use the following DDL statements for managing schemas:

- CREATE SCHEMA
- USE SCHEMA
- DROP SCHEMA

You must have `admin` privileges in HBase to run CREATE SCHEMA or DROP SCHEMA.

See the Apache Phoenix Grammar reference page for how you can use these DDL statements.

As you create physical tables, views, and indexes, you can associate them with a schema. If the schema already has namespace mapping enabled, the newly created objects automatically become part of the HBase namespace. The directory of the HBase namespace that maps to the Phoenix schema inherits the schema name. For example, if the schema name is `store1`, then the full path to the namespace is `$hbase.rootdir/data/store1`. See the "F.A.Q." section of Apache Phoenix Namespace Mapping for more information.

6.5.3. Associating Tables of a Schema to a Namespace

After you enable namespace mapping on a Phoenix schema that already has tables, you can migrate the tables to an HBase namespace. The namespace directory that contains the migrated tables inherits the schema name. For example, if the schema name is `store1`, then the full path to the namespace is `$hbase.rootdir/data/store1`.

System tables are migrated to the namespace automatically during the first connection after enabling namespace properties.

6.5.3.1. Associating in a Noncustomized Environment without Kerberos

Run the following command to associate a table:

```
$bin/psql.py {ZooKeeper_hostname -m schema_name.table_name}
```

6.5.3.2. Associating in a Customized Kerberos Environment

Prerequisite: In a Kerberos-secured environment, you must have admin privileges (user `hbase`) to complete the following task.

1. Navigate to the Phoenix home directory. The default location is `/usr/hdp/current/phoenix-client/`.
2. Run a command to migrate a table of a schema to a namespace, using the following command syntax for the options that apply to your environment:

   ```
   $ bin/psql.py {{ZooKeeper_hostnames:2181} | [:zookeeper.znode.parent] | [:HBase_headless_keytab_location] | [:principal_name] | [;TenantId=tenant_Id] | [;CurrentSCN=current_SCN]} -m {schema_name.table_name}
   ```

Additional information for valid command parameters:

ZooKeeper_hostnames	Enter the ZooKeeper hostname or hostnames that compose the ZooKeeper quorum. If you enter multiple hostnames, enter them as comma-separated values. This parameter is required. You must append the colon and ZooKeeper port number if you invoke the other security parameters in the command. The default port number is 2181.
zookeeper.znode.parent	This setting is defined in the `hbase-site.xml` file.
-m *schema_name.table_name*	The `-m` argument is required. There is a space before and after the `-m` option.

6.6. Phoenix Repair Tool

Important

The Phoenix repair tool of HDP is a **technical preview** and considered under development. Do not use this feature in your production systems. If you have questions regarding this feature, contact Support by logging a case on the Hortonworks Support Portal.

Apache Phoenix depends on the SYSTEM.CATALOG table for metadata information, such as table structure and index location, to function correctly. Use the Phoenix repair tool to validate the data integrity of the SYSTEM.CATALOG table. If a Phoenix client is not functioning as expected and throwing exceptions such as ArrayIndexOutOfBound or TableNotFound, this tool can help identify the problem and fix it.

The repair tool is designed to flag issues that are flagrant trouble spots and to fix SYSTEM.CATALOG problems in a way that does not radically affect your Phoenix system. The tool prompts you to confirm changes before the SYSTEM.CATALOG table is modified.

Do not use the Phoenix repair tool for an upgrade. The tool is designed to function ony with the current version of the system catalog and to use the HBase API directly.

6.6.1. Running the Phoenix Repair Tool

Tip

Run the HDFS **fsck** and HBase **hbck** tools *before* running the Phoenix repair tool. Checking the condition of HDFS and HBase is highly recommended because the Phoenix repair tool does not run on HDFS and HBase, both of which must be in working order for the repair tool to fix Phoenix problems.

About this Task

- The Phoenix repair tool looks for table records in the system catalog and collects all corresponding information about columns and indexes. If certain inconsistencies are detected, then the tool prompts you to verify that it should proceed with fixing the problems. The tool can fix the following problems:
  - Missing or disabled physical table
  - Incorrect number of columns in table metadata information
  - Table record has columns with an internal index that is out of range
- The tool performs a cross-reference check between user tables and indexes. If a user table has an index that misses a physical table, the tool offers to delete the link to this index as well as to delete the index table record from the system catalog. If the physical table is disabled, the tool asks whether it needs to be enabled.
- If you allow the Phoenix repair tool to fix an issue, the tool creates a snapshot of the SYSTEM.CATALOG table. The snapshot is created in case you want to rollback the repair operation.

Prerequisites

Verify that no concurrent execution of the Phoenix repair tool launches or runs while you run the tool. Also, ensure that no other clients modify the system catalog data while the tool runs.

Steps

1. Run the **psl.py** utility with the `-r` option:

```
/usr/hdp/current/phoenix-client/psql.py -r
```

2. If the tool detects previously stored snapshots on the system, respond to the Restore dialogue prompt:
   - Respond whether the tool should delete or retain the previously recorded snapshots.

- Indicate whether the tool should proceed with the integrity check or restore a table from the one of the snapshots.

Result and Next Step

After the tool completes the check, you can consider the SYSTEM.CATALOG table as validated. You can proceed with SQL operations in the Phoenix CLI.

7. Real-Time Data Analytics with Druid

Druid is an open-source data store designed for online analytical processing (OLAP) queries on event data. Druid supports the following data analytics features:

- Streaming data ingestion
- Real-time queries
- Scalability to trillions of events and petabytes of data
- Sub-second query latency

These traits make this data store particularly suitable for enterprise-scale business intelligence (BI) applications in environments that require minimal latency. With Druid you can have applications running interactive queries that "slice and dice" data in motion.

A common use case for Druid is to provide a data store that can return BI about streaming data that comes from user activity on a website or multidevice entertainment platform, from consumer events sent over by a data aggregator, or from any other large-scale set of relevant transactions or events from Internet-connected sources.

Druid is licensed under the Apache License, version 2.0.

7.1. Content Roadmap

The following table provides links to Druid information resources. The table points to resources that are not contained in this *HDP Data Access Guide.*

Important

The hyperlinks in the table and many others throughout this documentation jump to content published on the druid.io site. Do not download any Druid code from this site for installation in an HDP cluster. Instead, install by selecting Druid as a Service in an Ambari-asssisted HDP installation, as described in Installing and Configuring Druid.

Table 7.1. Druid Content Roadmap in Other Sources

Type of Information	Resources	Description
Introductions	About Druid (Source: druid.io)	Introduces the feature highlights of Druid, and explains in which environments the data store is best suited. The page also links to comparisons of Druid against other common data stores.
	Druid Concepts (Source: druid.io)	This page is the portal to the druid.io technical documentation. While the body of this page describes some of the main technical concepts and components, the right-side navigation pane outlines and links to the topics in the druid.io documentation.
	Druid: A Real-time Analytical Data Store (Source: druid.io)	This white paper describes the Druid architecture in detail, performance benchmarks, and an overview of Druid issues in a production

Type of Information	Resources	Description
		environment. The extensive References section at the end of the document point to a wide range of information sources.
Tutorial	Druid Quickstart (Source: druid.io)	A getting started tutorial that walks you through a Druid package, installation, and loading and querying data. The installation of this tutorial is for instructional purposes only and not intended for use in a Hortonworks Hadoop cluster.
Developing on Druid	Developing on Druid (Source: druid.io)	Provides an overview of major Druid components to help developers who want to code applications that use Druid-ingested data. The web page links to another about *segments*, which is an essential entity to understand when writing applications for Druid.
Data Ingestion	Batch Data Ingestion Loading Streams (Source: druid.io)	These two pages introduce how Druid can ingest data from both static files and real-time streams.
Queries of Druid Data	Querying (Source: druid.io)	Describes the method for constructing queries, supported query types, and query error messages.
Best Practices	Recommendations (Source: druid.io)	A list of tips and FAQs.

7.2. Architecture

Druid offers *streaming ingestion* and *batch ingestion* to support both of these data analytics modes. A Druid cluster consists of several Druid node types and components. Each Druid node is optimized to serve particular functions. The following list is an overview of Druid node types:

Realtime nodes ingest and index streaming data that is generated by system events. The nodes construct segments from the data and store the segments until these segments are sent to historical nodes. The realtime nodes do not store segments after the segments are transferred.

Historical nodes are designed to serve queries over immutable, historical data. Historical nodes download immutable, read-optimized Druid segments from deep storage and use memory-mapped files to load them into available memory. Each historical node tracks the segments it has loaded in ZooKeeper and transmits this information to other nodes of the Druid cluster when needed.

Broker nodes form a gateway between external clients and various historical and realtime nodes. External clients send queries to broker nodes. The nodes then break each query into smaller queries based on the location of segments for the queried interval and forwards them to the appropriate historical or realtime nodes. Broker nodes merge query results and send them back to the client. These nodes can also be configured to use a local or distributed cache for caching query results for individual segments.

Coordinator nodes mainly serve to assign segments to historical nodes, handle data replication, and to ensure that segments are distributed evenly across the historical nodes. They also provide a UI to manage different datasources and configure rules to load data and drop data for individual datas sources. The UI can be accessed via Ambari Quick Links.

Middle manager nodes are responsible for running various tasks related to data ingestion, realtime indexing, segment archives, etc. Each Druid task is run as a separate JVM.

Overlord nodes handle task management. Overlord nodes maintain a task queue that consists of user-submitted tasks. The queue is processed by assigning tasks in order to the middle manager nodes, which actually run the tasks. The overlord nodes also support a UI that provides a view of the current task queue and access to task logs. The UI can be accessed via Ambari Quick Links for Druid.

Tip

For more detailed information and diagrams about the architecture of Druid, see Druid: A Real-time Analytical Data Store Design Overview on druid.io.

7.3. Installing and Configuring Druid

Use Apache Ambari to install Druid. Manual installation of Druid to your HDP cluster is not recommended.

After you have a running Ambari server set up for HDP, you can install and configure Druid for your Hadoop cluster just like any other Ambari Service.

Important

You must use Ambari 2.5.0 or a later version to install and configure Druid. Earlier versions of Ambari do not support Druid as a service. Also, Druid is available as an Ambari Service when you run with the cluster with HDP 2.6.0 and later versions.

If you need to install and start an Ambari server, see the Getting Ready section of the *Apache Ambari Installation Guide*.
If a running Ambari server is set up for HDP, see Installing, Configuring, and Deploying a HDP Cluster **and** the following subsections. While you can install and configure Druid for the Hadoop cluster just like any other Ambari Service, the following subsections contain Druid-specific details for some of the steps in the Ambari-assisted installation.

7.3.1. Interdependencies for the Ambari-Assisted Druid Installation

To use Druid in a real-world environment, your Druid cluster must also have access to other resources to make Druid operational in HDP:

- *ZooKeeper*: A Druid instance requires installation of Apache ZooKeeper. Select **ZooKeeper** as a Service during Druid installation. If you do not, the Ambari installer does not complete. ZooKeeper is used for distributed coordination among Druid nodes and for leadership elections among coordinator and overlord nodes.

- *Deep storage:* HDFS or Amazon S3 can be used as the deep storage layer for Druid in HDP. In Ambari, you can select **HDFS** as a Service for the deep storage of data. Aternatively, you can set up Druid to use Amazon S3 as the deep storage layer by

setting the *druid.storage.type* property to **s3**. The cluster relies on the distributed filesystem to store Druid segments so that there is permanent backup of the data.

- *Metadata storage*: The metadata store is used to persist information about Druid segments and tasks. MySQL, Postgres, and Derby are supported metadata stores. You have the opportunity to select the metadata database when you install and configure Druid with Ambari.
- *Batch execution engine:* Select **YARN + MapReduce2** as the appropriate execution resource manager and execution engine, respectively. Druid **hadoop index** tasks use MapReduce jobs for distributed ingestion of large amounts of data.
- *(Optional) Druid metrics reporting*: If you plan to monitor Druid performance metrics using Grafana dashboards in Ambari, select **Ambari Metrics System** as a Service.

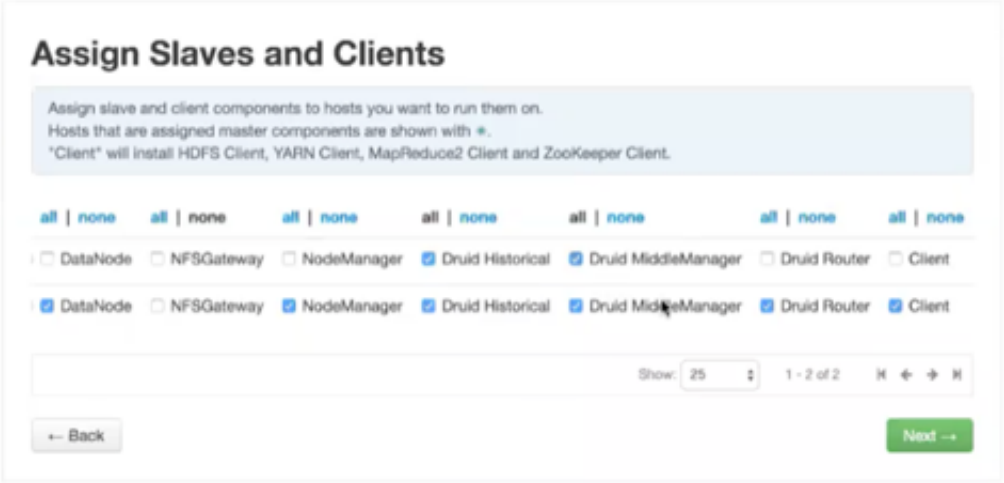
Tip

If you plan to deploy high availability (HA) on a Druid cluster, review the High Availability in Druid Clusters section below to learn what components to install and how to configure the installation so that the Druid instance is primed for a HA environment.

7.3.2. Assigning Slave and Client Components

On the Assign Slaves and Clients window of Ambari, generally you should select **Druid Historical** and **Druid MiddleManager** for multiple nodes. (You may also need to select other components that are documented in the *Apache Ambari Installation Guide*.)The purpose of these components are as follows:

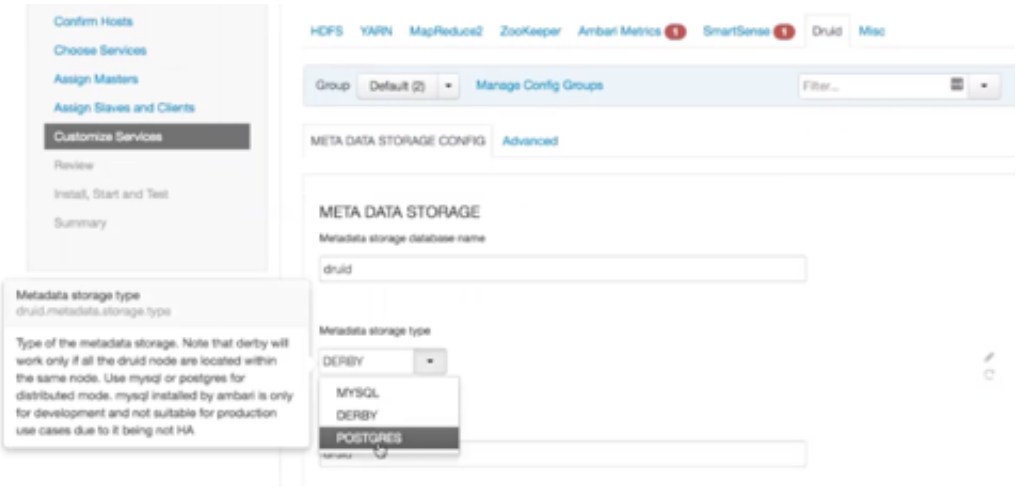
Druid Historical: Loads data segments.
Druid MiddleManager: Runs Druid indexing tasks.



7.3.3. Configuring the Druid Installation

Use the Customize Services window of Ambari installer to finalize the configuration of Druid.

Select **Druid** > **Metadata storage type** to access the drop-down menu for choosing the metadata storage database. When you click the drop-down menu, notice the tips about the database types that appear in the GUI.



To proceed with the installation after selecting a database, enter your **`admin`** password in the **Metadata storage password** fields. If MySQL is your metadata storage database, also follow the steps in the Setting up MySQL for Druid section below.

Toggle to the **Advanced** tab. Ambari has Stack Advisor, which is configuration wizard. Stack Advisor populates many configuration settings based on the your previously entered settings and your environment. Although Stack Advisor provides many default settings for Druid nodes and other related entities in the cluster, you might need to manually tune the configuration parameter settings. Review the following druid.io documentation to determine if you need to change any default settings and how other manual settings operate:

- Configuring Druid (Common Properties)
- Runtime Configuration
- Coordinator Node Configuration
- Historical Node Configuration
- Broker Node Configuration
- Realtime Node Configuration
- Logging

7.3.3.1. Setting up MySQL for Druid

Complete the following task only if you chose MySQL as the metadata store for Druid.

1. On the Ambari Server host, stage the appropriate MySQL connector for later deployment.

   a. Install the connector.

RHEL/CentOS/Oracle Linux

```
yum install mysql-connector-java*
```

SLES

```
zypper install mysql-connector-java*
```

Debian/Ubuntu

```
apt-get install libmysql-java
```

b. Confirm that `mysql-connector-java.jar` is in the Java share directory.

```
ls /usr/share/java/mysql-connector-java.jar
```

c. Make sure the Linux permission of the `.JAR` file is **644**.

d. Execute the following command:

```
ambari-server setup --jdbc-db=mysql --jdbc-driver=/usr/share/
java/mysql-connector-java.jar
```

2. Create the Druid database.

- The Druid database must be created prior using the following command:

```
# mysql -u root -p

CREATE DATABASE <DRUIDDATABASE> DEFAULT CHARACTER SET utf8;
```

replacing <DRUIDDATABASE> with the Druid database name.

3. Create a Druid user with sufficient superuser permissions.

- Enter the following in the MySQL database admin utility:

```
# mysql -u root -p

CREATE USER '<DRUIDUSER>'@'%' IDENTIFIED BY '<DRUIDPASSWORD>';

GRANT ALL PRIVILEGES ON <DRUIDDATABASE>.* TO '<DRUIDUSER>'@'%’;

FLUSH PRIVILEGES;
```

replacing <DRUIDUSER> with the Druid user name and <DRUIDPASSWORD> with the Druid user password.

7.4. Security and Druid

Important

Place the Druid endpoints behind a firewall. More robust security features that remove the need to deploy a firewall around Druid are in development.

You can configure Druid nodes to integrate with a Kerberos-secured Hadoop cluster to enable authentication between Druid and other HDP Services. You can enable the authentication with Ambari 2.5.0+, which lets you secure the HTTP endpoints by including a SPNEGO-based Druid extension. The mechanism by which Kerberos security uses keytabs and principals to strengthen security is described in Kerberos Overview and Kerberos Principals.

A benefit of enabling authentication in this way is that it can connect Druid Web UIs to the core Hadoop cluster while maintaining Kerberos protection. The main Web UIs to use with HDP are the Coordinator Console and the Overlord Console. See Coordinator Node and Overlord Node on the druid.io documentation website for more information about the consoles for these nodes.

7.4.1. Securing Druid Web UIs and Accessing Endpoints

Enabling SPNEGO-based Kerberos authentication between the Druid HTTP endpoints and the rest of the Hadoop cluster requires running the Ambari Kerberos Wizard and manually connecting to Druid HTTP endpoints in a command line. After authenticating successfully to Druid, you can submit queries through the endpoints.

Procedure 7.1. Enabling Kerberos Authentication in Druid

Prerequisite: The Ambari Server and Services on the cluster must have SPNEGO-based Kerberos security enabled. See the *Ambari Security Guide* if you need to configure and enable SPNEGO-based Kerberos authentication.

Important

The whole HDP cluster is down after you configure the Kerberos settings and initialize the Kerberos wizard in the following task. Ensure you can have temporary down-time before completing all steps of this task.

1. Follow the steps in Launching the Kerberos Wizard (Automated Setup) until you get to the **Configure Identities** window of the Ambari Kerberos Wizard. Do not click the **Next** button until you adjust advanced Druid configuration settings as follows:

2. On the **Configure Identities** window:

   a. Review the principal names, particularly the Ambari Principals on the **General** tab. These principal names, by default, append the name of the cluster to each of the Ambari principals. You can leave the default appended names or adjust them by removing the `-cluster-name` from the principal name string. For example, if your cluster is named `druid` and your realm is **`EXAMPLE.COM`**, the Druid principal that is created is **`druid@EXAMPLE.COM`**.

   b. Select the **Advanced** tab > **Druid** drop-down menu.

   c. Use the following table to determine for which Druid properties, if any, you need to change the default settings. In most cases, you should not need to change the default values.

Table 7.2. Advanced Druid Identity Properties of Ambari Kerberos Wizard

Property	Default Value Setting	Description
druid.hadoop.security.spnego.excludedPaths	**['status']** If you want to set more than one path, enter values in the following format: **['/status','/condition']**	Specify here HTTP paths that do not need to be secured with authentication. A possible use case for providing paths here are to test scripts outside of a production environment.
druid.hadoop.security.spnego.keytab	***keytab_dir*/spnego.service.keytab**	This is the SPNEGO service keytab that is used for authentication.
druid.hadoop.security.spnego.principal	**HTTP/*_HOST@realm***	This is the SPNEGO service principal that is used for authentication.
druid.security.extensions.loadlist	**[druid-kerberos]**	This indicates the Druid security extension to load for Kerberos.

d. Confirm your configuration. Optionally, you can download a CSV file of the principals and keytabs that Ambari can automatically create.

e. Click **Next** to proceed with kerberization of the cluster. During the process, all running Services are stopped. Kerberos configuration settings are applied to various components, and keytabs and principals are generated. When the Kerberos process finishes, all Services are restarted and checked.

Tip

Initializing the Kerberos Wizard might require a significant amount of time to complete, depending on the cluster size. Refer to the GUI messaging on the screen for progress status.

Procedure 7.2. Accessing Kerberos-Protected HTTP Endpoints

Before accessing any Druid HTTP endpoints, you need to authenticate yourself using Kerberos and get a valid Kerberos ticket as follows:

1. Log in via the Key Distribution Center (KDC) using the following **`kinit`** command, replacing the arguments with your real values:

```
kinit -k -t keytab_file_path user@REALM.COM
```

2. Verify that you received a valid Kerberos authentication ticket by running the **`klist`** command. The console prints a **`Credentials cache`** message if authentication was successful. An error message indicates that the credentials are not cached.

3. Access Druid with a **`curl`** command. When you run the command, you must include the SPNEGO protocol *`--negotiate`* argument. (Note that this argument has double

hyphens.) The following is an *example* command. Replace **anyUser**, **cookies.txt**, and **endpoint** with your real values.

```
curl --negotiate -u:anyUser -b ~/cookies.txt -c ~/cookies.txt -X POST -
H'Content-Type: application/json' http://_endpoint
```

4. Submit a query to Druid in the following *example* **curl** command format:

```
curl --negotiate -u:anyUser -b ~/cookies.txt -c ~/cookies.txt -X POST
 -H'Content-Type: application/json'  http://broker-host:port/druid/v2/?
pretty -d @query.json
```

7.5. High Availability in Druid Clusters

Ambari provides the ability to configure the High Availability (HA) features available in Druid and other HDP Stack Services.

7.5.1. Configuring Druid Clusters for High Availability

HA by its nature requires a multinode structure with somewhat more sophisticated configuration than a cluster without HA. Do not use local storage in an HA environment.

7.5.1.1. Configure a Cluster with an HDFS Filesystem

Prerequisites

MySQL or Postgres must be installed as the metadata storage layer. Configure your metadata storage for HA mode to avoid outages that impact cluster operations. See your database documentation. Derby does not support a multinode cluster with HA.
At least three ZooKeeper nodes must be dedicated to HA mode.

Steps

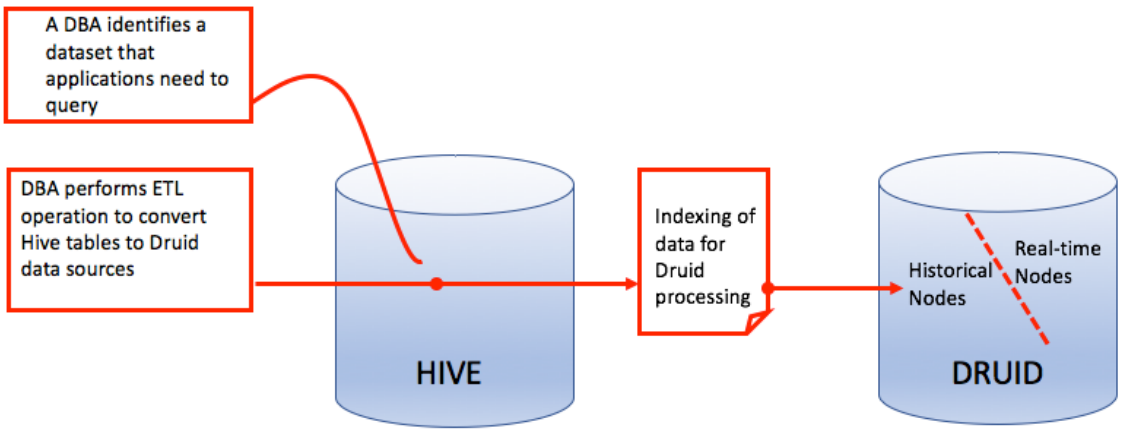
1. Enable Namenode HA using the wizard as described in Configuring NameNode High Availability
2. Install the Druid Overlord, Coordinator, Broker, Realtime, and Historical processes *on multiple nodes that are distributed among different hardware servers.*
   - Within each Overlord and Coordinator domain, ZooKeeper determines which node is the Active node. The other nodes supporting each process are in Standby state until an Active node stops running and the Standby nodes receive the failover.
   - Multiple Historical and Realtime nodes also serve to support a failover mechanism. But for Broker and Realtime processes, there are no designated Active and Standby nodes.
   - Muliple instances of Druid Broker processes is required for HA. *Recommendations:* Use an external, virtual IP address or load balancer to direct user queries to multiple Druid Broker instances. A Druid Router can also serve as a mechanism to route queries to multiple broker nodes.
3. Ensure that the replication factor for each datasource is set greater than **1** in the Coordinator process rules. If no datasource rule configurations were changed, no action is required because the default value is **2**.

7.6. Leveraging Druid to Accelerate Hive SQL Queries

HDP integration of Hive with Druid places a SQL layer on Druid so that applications can efficiently perform analytic queries on both real-time and historical data. After Druid ingests data from a Hive enterprise data warehouse (EDW), the interactive and sub-second query capabilities of Druid can be used to accelerate queries on historical data from the EDW. Hive integration with Druid enables applications such as Tableau to scale while queries run concurrently on both real-time and historical data.

The following figure is an overview of how Hive historical data can be brought into a Druid environment. Queries analyzing Hive-sourced data are run directly on the historical nodes of Druid after indexing between the two databases completes.

Figure 7.1. Batch Ingestion of Hive Data into Druid



Important

If Kerberos is enabled on your cluster, then the only way to query Druid datasources that are imported from Hive is to use Hive LLAP daemons. Ensure that the Hive property is set as hive.llap.execution.mode=all.

7.6.1. How Druid Indexes Hive-Sourced Data

Before you can create Druid datasources using Hive data as your source, you must understand how data of a Hive external table map is mapped to the column orientation and segment files of Druid.

Each Druid segment consists of the following objects to facilitate fast lookup and aggregation:

Timestamp column — The SQL-based timestamp column is filled in based on how you set the time granularity of imported Hive data and what time

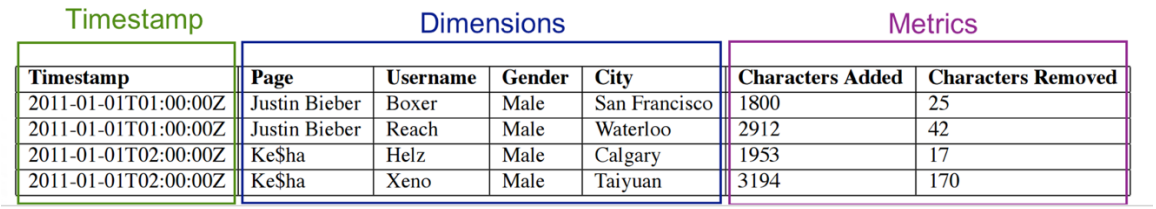
range of data is selected in the Hive external table. This column is essential for indexing the data in Druid because Druid itself is a time-series database. The timestamp column must be named **`__time`**.

Dimension columns — The dimension columns are used to set string attributes for search and filter operations. To index a Hive-sourced column as a Druid dimension column, you must cast the column as a string type.

Metric columns — Metric columns are used to index metrics that are supposed to be used as aggregates or measures. To index a Hive-sourced column as a Druid metric column, you must cast the column as a Hive numeric data type.

The following figure represents an example of how Druid data can be categorized in the three column types.

Figure 7.2. Example of Column Categorization in Druid



Timestamp	Page	Username	Gender	City	Characters Added	Characters Removed
2011-01-01T01:00:00Z	Justin Bieber	Boxer	Male	San Francisco	1800	25
2011-01-01T01:00:00Z	Justin Bieber	Reach	Male	Waterloo	2912	42
2011-01-01T02:00:00Z	Ke$ha	Helz	Male	Calgary	1953	17
2011-01-01T02:00:00Z	Ke$ha	Xeno	Male	Taiyuan	3194	170

7.6.2. Transforming Hive Data to Druid Datasources

About this Task

A DBA runs a Hive SQL command invoking the Druid storage handler, specifies the Druid segment granularity, and maps selected Hive columns to Druid column types.

Steps

1. Put all the Hive data to undergo ETL in an external table.
2. Run a CREATE TABLE AS SELECT statement to create a new Druid datasource. The following is an example of a statement pushing Hive data to Druid.

   Example:

```
CREATE TABLE ssb_druid_hive
STORED BY 'org.apache.hadoop.hive.
druid.DruidStorageHandler'
TBLPROPERTIES (
  "druid.segment.granularity" = "MONTH",
  "druid.query.granularity" = "DAY")
AS
SELECT
  cast(d_year || '-' || d_monthnuminyear || '-' || d_daynuminmonth as
 timestamp) as `__time`,
  cast(c_city as string) c_city,
  cast(c_nation as string) c_nation,
  cast(c_region as string) c_region,
  cast(d_weeknuminyear as string) d_weeknuminyear,
```

```
  cast(d_year as string) d_year,
  cast(d_yearmonth as string) d_yearmonth,
  cast(d_yearmonthnum as string) d_yearmonthnum,
  cast(lo_discount as string) lo_discount,
  cast(lo_quantity as string) lo_quantity,
  cast(p_brand1 as string) p_brand1,
  cast(p_category as string) p_category,
  cast(p_mfgr as string) p_mfgr,
  cast(s_city as string) s_city,
  cast(s_nation as string) s_nation,
  cast(s_region as string) s_region,
  lo_revenue,
  lo_extendedprice * lo_discount discounted_price,
  lo_revenue - lo_supplycost net_revenue
FROM
  ssb_10_flat_orc.customer, ssb_10_flat_orc.dates, ssb_10_flat_orc.
lineorder,
  ssb_10_flat_orc.part, ssb_10_flat_orc.supplier
where
  lo_orderdate = d_datekey and lo_partkey = p_partkey
  and lo_suppkey = s_suppkey and lo_custkey = c_custkey;
```

Explanation of Preceding SQL Statement

The following breakdown of the preceding SQL statement explains the main elements of a statement that can transform Hive data into a time series-based Druid datasource. Values *in italic* are only examples, which should be replaced by your data warehouse and analytics parameters as needed.

`CREATE TABLE` *`ssb_druid_hive`*	Creates the Hive external table and assigns a name to it. You must use a table name that is not already used by another Druid datasource.
`STORED BY, 'org.apache.hadoop.hive., druid.DruidStorageHandler'`	This calls the Druid storage handler so that the Hive data can be transformed to a Druid datasource.
`TBLPROPERTIES ("druid.segment.granularity" = "`*`MONTH`*`", "druid.query.granularity" = "`*`DAY`*`")`	
`AS SELECT cast(`*`d_year`*` || '-' || `*`d_monthnuminyear`*` || '-' || `*`d_daynuminmonth`*` as timestamp) as `__time`,`	Creates the **`__time`** column, which is a required SQL timestamp column for the Druid datasource.
`cast(c_`*`city`*` as string) c_`*`city`*`, cast(c_`*`nation`*` as string) c_`*`nation`*`, cast(c_`*`region`*` as string) c_`*`region`*`, cast(d_`*`weeknuminyear`*` as`	**`cast (… as string)`** statements index columns as dimensions. Dimensions in the Druid datasource are used to search and filter.

`string) d_weeknuminyear,` `etc.`	
`lo_extendedprice * lo_discount discounted_price, lo_revenue - lo_supplycost net_revenue`	These lines preaggregate metrics columns. To index a column as metrics, you need to cast the column to a Hive numeric data type.
`FROM ssb_10_flat_orc.customer, ssb_10_flat_orc.dates,` etc.	The numeric value indicates the data scale. The other information corresponds with the name and other components of the source Hive tables.

Table 7.3. Explanation of Table Properties for Hive to Druid ETL

Table Property	Required?	Description	Valid Values
druid.segment.popularity	No	Defines how the data is physically partitioned. The values that are permissible here correspond with Druid segment granularity.	"YEAR", "MONTH", "WEEK", "DAY", "HOUR", "MINUTE", "SECOND"
druid.query.granularity	No	Defines how much granularity to store in a segment. The values that are permissible here correspond with Druid query granularity.	"YEAR", "MONTH", "WEEK", "DAY", "HOUR", "MINUTE", "SECOND"

Next Steps

If you need Druid to ingest Hive data that follows the same schema as the first data set that you transformed, you can do so with the INSERT INTO statement.

7.6.3. Performance-Related `druid.hive.*` Properties

The following table lists some of the Druid properties that can be used by Hive. These properties can affect performance of your Druid-Hive integration.

If Hive and Druid are installed with Ambari, the properties are set and tuned for your cluster automatically. However, you may need to fine-tune some properties if you detect performance problems with applications that are running the queries The table below is primarily for HDP administrators who need to troubleshoot and customize a Hive-Druid integration.

Table 7.4. Performance-Related Properties

Property	Description
hive.druid.indexer.segments.granularity	Granularity of the segments created by the Druid storage handler.
hive.druid.indexer.partition.size.max	Maximum number of records per segment partition.
hive.druid.indexer.memory.rownum.max	Maximum number of records in memory while storing data in Druid.
hive.druid.broker.address.default	Address of the Druid broker node. When Hive queries Druid, this address must be declared.

Property	Description
hive.druid.coordinator.address.default	Address of the Druid coordinator node. It is used to check the load status of newly created segments.
hive.druid.select.threshold	When a SELECT query is split, this is the maximum number of rows that Druid attempts to retrieve.
hive.druid.http.numConnection	Number of connections used by the HTTP client.
hive.druid.http.read.timeout	Read timeout period for the HTTP client in ISO8601 format. For example, **`P2W, P3M, PT1H30M, PT0.750S`** are possible values.
hive.druid.sleep.time	Sleep time between retries in ISO8601 format.
hive.druid.basePersistDirectory	Local temporary directory used to persist intermediate indexing state.
hive.druid.storage.storageDirectory	Deep storage location of Druid.
hive.druid.metadata.base	Default prefix for metadata table names.
hive.druid.metadata.db.type	Metadata database type. The only valid values are "mysql" and "postgresql"
hive.druid.metadata.uri	URI to connect to the database.
hive.druid.working.directory	Default HDFS working directory used to store some intermediate metadata.
hive.druid.maxTries	Maximum number of retries to connect to Druid before throwing an exception.
hive.druid.bitmap.type	Encoding algorithm use to encode the bitmaps.

Tip

If you installed both Hive and Druid with Ambari, then do not change any of the `hive.druid.*` properties other than those above when there are performance issues.