Data Access 3

Apache Hive overview

Date of Publish: 2018-07-12

http://docs.hortonworks.com

Contents

What's new in this release: Apache Hive

HDP 3.0.1 includes partitioning of materialized views, which can improve query responsiveness and maintenance fixes.

HDP 3.0 includes Apache Hive 3 enhancements that can help you improve query performance and comply with regulations. The following list briefly describes a few key enhancements of HDP 3.0 and covers unsupported interfaces.

Workload management

Using workload management, you can configure who uses resources, how much can be used, and how quickly Hive responds to resource requests. Managing resources is critical to Hive LLAP (low-latency analytical processing), especially in a multitenant environment. Using workload management, you can create resource pools and allocate resources to match availability needs and prevent contention for those resources. Workload management improves parallel query execution and cluster sharing for queries running on Hive LLAP, and also improves performance of non-LLAP queries. Workload management reduces resource starvation in large clusters. You implement workload management on the command line using Hive.

Transaction processing improvements

Mature versions of ACID (Atomicity, Consistency, Isolation, and Durability) transaction processing and LLAP evolve in Hive and HDP 3.0. ACID tables are enhanced to serve as the default table type in HDP 3.0, without performance or operational overload. LLAP processes queries in subseconds. Using ACID table operations facilitates compliance with the right to be forgotten requirement of the GDPR (General Data Protection Regulation). Application development and operations are simplified with stronger transactional guarantees and simpler semantics for SQL commands. You do not need to bucket ACID tables, so maintenance is easier. You no longer need to perform ACID delete operations in a Hive table.

Materialized views

With improvements in transactional semantics comes advanced optimizations, such as materialized view rewrites and automatic query cache. With these optimizations, you can deploy new Hive application types. Because multiple queries frequently need the same intermediate roll up or joined table, you can avoid costly, repetitious query portion sharing, by precomputing and caching intermediate tables into views. The query optimizer automatically leverages the precomputed cache, improving performance. Materialized views increase the speed of join and aggregation queries in business intelligence (BI) and dashboard applications, for example.

Cost-based optimizer enhancements

Hive can push down the filtering, sorting, and joining of columns in a query. For example, MySQL tables joins can be pushed down to underlying database.

Direct, low latency Hive query of Kafka topics

You can ingest Kafka into ACID tables, or query the data in the Kafka message from Hive. With HDP 3.0, you can create a Druid table within Hive from a Kafka topic in a single command. This feature simplifies queries of Kafka data by eliminating the data processing step between delivery by Kafka and querying in Druid.

Superset

HDP 3 introduces a technical preview of Apache Superset, the data exploration and visualization UI platform. Superset is a way to create HDP dashboards. Using Superset, installed by default as a service in Ambari, you can connect to Hive, create visualizations of Hive data, and create custom dashboards on Hive datasets. Superset is an alternative to Hive View, which is not available in HDP 3.0.

Spark integration with Hive

You can use Hive 3 to query data from Apache Spark and Apache Kafka applications, without workarounds. The Hive Warehouse Connector supports reading and writing Hive tables from Spark.

Hive security improvements

Apache Ranger secures Hive data by default. Through the Hive Warehouse Connector (HWC), you can secure data access at the column or row level from Spark. To meet customer demands for concurrency improvements, ACID support for GDPR (General Data Protection Regulation), render security, and other features, Hive now tightly controls the file system and computer memory resources. With the additional control, Hive better optimizes workloads in shared files and YARN containers. The more Hive controls the file system, the better Hive can secure data.

Query result cache

Hive filters and caches similar or identical queries. Hive does not recompute the data that has not changed. Caching repetitive queries can reduce the load substantially when hundreds or thousands of users of BI tools and web services query Hive.

Information schema database

Hive creates two databases from JDBC data sources when you add the Hive service to a cluster: information_schema and sys. All Metastore tables are mapped into your tablespace and available in sys. The information_schema data reveals the state of the system, similar to sys database data. You can query information_schema using SQL standard queries, which are portable from one DBMS to another.

Deprecated, unavailable, and unsupported interfaces

In HDP 3.0 and later, Hive does not support the following features:

- Apache Hadoop Distributed Copy (DistCp)
- WebHCat
- Hcat CLI
- Hive CLI (replaced by Beeline)
- SQL Standard Authorization
- MapReduce execution engine (replaced by Tez)

Related Information

Workload management

Hive 3 ACID transactions

Using materialized views

Visualizing Apache Hive data using Apache Superset

HiveWarehouseConnector for handling Apache Spark data

Apache Hive 3 architectural overview

Apache Hive 3 architectural overview

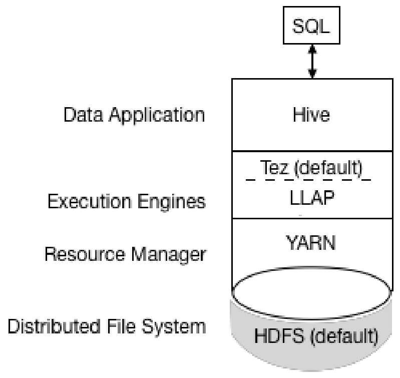
Understanding Apache Hive 3 major design changes, such as default ACID transaction processing and support for only the thin hive client, can help you use new features to address the growing needs of enterprise data warehouse systems.

Execution engine changes

Apache Tez replaces MapReduce as the default Hive execution engine. MapReduce is no longer supported, and Tez stability is proven. With expressions of directed acyclic graphs (DAGs) and data transfer primitives, execution of Hive queries under Tez improves performance. SQL queries you submit to Hive are executed as follows:

- Hive compiles the query.
- Tez executes the query.
- YARN allocates resources for applications across the cluster and enables authorization for Hive jobs in YARN queues.
- Hive updates the data in HDFS or the Hive warehouse, depending on the table type.
- Hive returns query results over a JDBC connection.

A simplified view of this process is shown in the following figure:



If a legacy script or application specifies MapReduce for execution, an exception occurs. Most user-defined functions (UDFs) require no change to execute on Tez instead of MapReduce.

Design changes that affect security

The following Hive 3 architectural changes provide improved security:

- Tightly controlled file system and computer memory resources, replacing flexible boundaries: Definitive boundaries increase predictability. Greater file system control improves security.
- Optimized workloads in shared files and YARN containers

By default, the HDP 3.0 Ambari installation adds Apache Ranger security services. The major authorization model in Hive is Ranger. This model permits only Hive to access HDFS. Hive enforces access controls specified in Ranger. This model offers stronger security than other security schemes and more flexibility in managing policies.

If you do not enable the Ranger security service, or other security, by default Hive uses storage-based authorization (SBA) based on user impersonation.

HDFS permission changes

In HDP 3.0, SBA relies heavily on HDFS access control lists (ACLs). ACLs are an extension to the permissions system in HDFS. HDP 3.0 turns on ACLs in HDFS by default, providing you with the following advantages:

- Increased flexibility when giving multiple groups and users specific permissions
- Convenient application of permissions to a directory tree rather than by individual files

Transaction processing changes

You can deploy new Hive application types by taking advantage of the following improvements in transaction processing:

- Mature versions of ACID transaction processing and LLAP:

  ACID tables are the default table type in HDP 3.0.

  ACID enabled by default causes no performance or operational overload.
- Simplified application development, operations with stronger transactional guarantees, and simpler semantics for SQL commands

  You do not need to bucket ACID tables in HDP 3.0, so maintenance is easier.
- Materialized view rewrites
- Automatic query cache
- Advanced optimizations

Hive client changes

Hive 3 supports only the thin client Beeline for running queries and Hive administrative commands from the command line. Beeline uses a JDBC connection to HiveServer to execute all commands. Parsing, compiling, and executing operations occur in HiveServer. Beeline supports the same command-line options as the Hive CLI with one exception: Hive Metastore configuration changes.

You enter supported Hive CLI commands by invoking Beeline using the hive keyword, command option, and command. For example, hive -e set. Using Beeline instead of the thick client Hive CLI, which is no longer supported, has several advantages, including the following:

- Instead of maintaining the entire Hive code base, you now maintain only the JDBC client.
- Startup overhead is lower using Beeline because the entire Hive code base is not involved.

A thin client architecture facilitates securing data in these ways:

- Session state, internal data structures, passwords, and so on reside on the client instead of the server.
- The small number of daemons required to execute queries simplifies monitoring and debugging.

HiveServer enforces whitelist and blacklist settings of Hive configuration properties. Beeline does not support hive -e set key=value to configure the Hive Metastore. Using the blacklist, you can restrict memory configuration to prevent HiveServer instability. You can configure multiple HiveServer instances with different whitelists and blacklists to establish different levels of stability.

The change in Hive client requires you to use the grunt command line to work with Apache Pig.

Apache Hive Metastore changes

Hive now uses a remote metastore instead of a metastore embedded in the same JVM instance as the Hive service; consequently, Ambari no longer starts the metastore using hive.metastore.uris=' '. The Hive metastore resides on a node in a cluster managed by Ambari as part of the HDP stack. A standalone server outside the cluster is not supported. You no longer set key=value commands on the command line to configure Hive Metastore. You configure properties in hive-site.xml. The Hive catalog resides in Hive Metastore, which is RDBMS-based as it was in earlier releases. Using this architecture, Hive can take advantage of RDBMS resources in a cloud deployments.

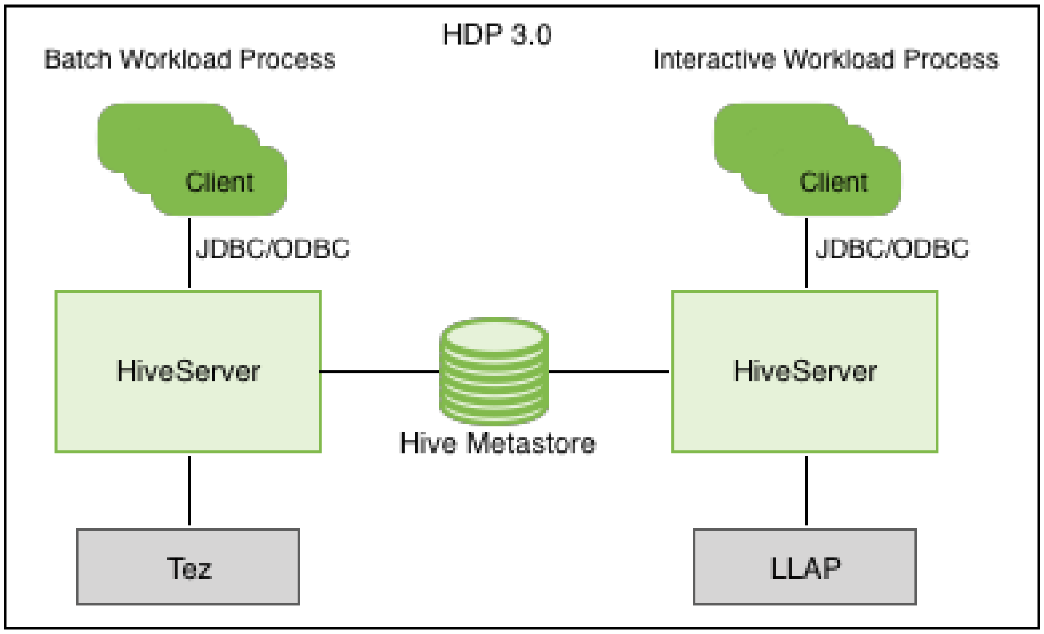
Spark catalog changes

Spark and Hive now use independent catalogs for accessing SparkSQL or Hive tables on the same or different platforms. A table created by Spark resides in the Spark catalog. A table created by Hive resides in the Hive catalog. Although independent, these tables interoperate.

You can access ACID and external tables from Spark using the HiveWarehouseConnector.

Query execution of batch and interactive workloads

The following diagram shows the HDP 3.0 query execution architecture for batch and interactive workloads:



You can connect to Hive using a JDBC command-line tool, such as Beeline, or using an JDBC/ODBC driver with a BI tool, such as Tableau. Clients communicate with an instance of the same HiveServer version. You configure the settings file for each instance to perform either batch or interactive processing.

Apache Hive 3 upgrade process

Supplemental information about preparing for an upgrade, upgrading, and using Hive tables after upgrading to Hive 3 helps you achieve a successful HDP and Apache Ambari major upgrade.

Some transactional tables require a major compaction before upgrading to 3.0. Running the Hive pre-upgrade tool identifies the tables that need such a compaction and provides scripts that you run to perform the compaction. Depending on the number of tables and partitions, and the amount of data involved, compactions might take a significant amount of time and resources. The script output of the pre-upgrade tool includes some heuristics that might help estimate the time required. If no script is produced, no compaction is needed.

Compaction cannot occur if the pre-upgrade tool cannot connect to Hive Metastore. During compaction, shutting down HiveServer2 is recommended to prevent users from executing any update, delete, or merge statements on tables during compaction and for the duration of the upgrade process.

You should run the pre-upgrade tool command on the command line after upgrading Ambari 2.6.2.2 to 2.7.x. You do not actually use Ambari to run this command.

The following properties can affect compaction:

- hive.compactor.worker.threads

  Specifies limits of concurrent compactions.
- hive.compactor.job.queue

  Specifies the Yarn queue of compaction jobs. Each compaction is a MapReduce job.

The pre-upgrade tool looks for files in an ACID table that contains update or delete events, and generates scripts to compact these tables. You prepare Hive for upgrade to obtain and run the scripts. Assuming you upgraded Ambari at some point, you can then upgrade HDP components, including Hive. After upgrading, check generated logs for any errors. Check that the upgrade process correctly converted your tables.

Related Information

Prepare Hive for Upgrade

Prepare Hive for Upgrade: IBM Power Systems
Hive Post-upgrade tasks
Drop an external table along with data

Changes after upgrading to Apache Hive 3

To locate and use your Apache Hive 3 tables after an upgrade, you need to understand the changes that occur during the upgrade process. Changes to the management and location of tables, permissions to HDFS directories, table types, and ACID-compliance occur.

Hive Management of Tables

Hive 3 takes more control of tables than Hive 2, and requires managed tables adhere to a strict definition. The level of control Hive takes over tables is similar to that of a traditional data base. If there's a change to the Hive data, hive knows about it. This control is a required framework for performtance features. For example, if Hive knows that resolving a query does not require scanning tables for new data, Hive returns results from the hive query result cache.

When the underlying data in a materialized view changes, Hive needs to rebuild the materialized view. ACID properties reveal exactly which rows changed, and only those need to be processed and added to the materialized view.

Hive changes to ACID properties

Hive 2.x and 3.x have transactional and non-transactional tables. Transactional tables have atomic, consistent, isolation, and durable (ACID) properties. In Hive 2.x, the initial version of ACID transaction processing is ACID v1. In Hive 3.x, the mature version of ACID is ACID v2, which is the default table type in HDP 3.0.

Native and non-native storage formats

Storage formats are a factor in upgrade changes to table types. Hive 2.x and 3.x supports the following Hadoop native and non-native storage formats:

- Native: Tables with built-in support in Hive, such as those in the following file formats:
  - Text
  - Sequence File
  - RC File
  - AVRO File
  - ORC File
  - Parquet File
- Non-native: Tables that use a storage handler, such as the DruidStorageHandler or HBaseStorageHandler

HDP 3.x upgrade changes to table types

The following table compares Hive table types and ACID operations before an upgrade from HDP 2.x and after an upgrade to HDP 3.x. The ownership of the Hive table file is a factor in determining table types and ACID operations after the upgrade.

Table 1: HDP 2.x and 3.x Table Type Comparison

HDP 2.x				**HDP 3.x**	
Table Type	**ACID v1**	**Format**	**Owner (user) of Hive Table File**	**Table Type**	**ACID v2**
External	No	Native or non-native	hive or non-hive	External	No
Managed	Yes	ORC	hive or non-hive	Managed, updatable	Yes

HDP 2.x				**HDP 3.x**	
Table Type	**ACID v1**	**Format**	**Owner (user) of Hive Table File**	**Table Type**	**ACID v2**
Managed	No	ORC	hive	Managed, updatable	Yes
			non-hive	External, with data delete*	No
Managed	No	Native (but non-ORC)	hive	Managed, insert only	Yes
			non-hive	External, with data delete*	No
Managed	No	Non-native	hive or non-hive	External, with data delete*	No

* See Dropping an External Table Along with the Data (link below).

Hive Impersonation and Security Changes

Hive impersonation was enabled by default in Hive 2 (doAs=true), and disabled by default in Hive 3. Hive impersonation runs Hive as end user, or not. Ranger is recommended for use with Hive 3. You can control HDFS security using Ranger policies, which is simpler than setting up permissions.

Other HDP 3.x upgrade changes

Managed, ACID tables that are not owned by the hive user remain managed tables after the upgrade, but hive becomes the owner.

After the upgrade, the format of a Hive table is the same as before the upgrade. For example, native or non-native tables remain native or non-native, respectively.

After the upgrade, the location of managed tables or partitions do not change under any one of the following conditions:

- The old table or partition directory was not in its default location /apps/hive/warehouse before the upgrade.
- The old table or partition is in a different file system than the new warehouse directory.
- The old table or partition directory is in a different encryption zone than the new warehouse directory.

Otherwise, the location of managed tables or partitions does change: The upgrade process moves managed files to /warehouse/tablespace/managed/hive. By default, Hive places any new external tables you create in HDP 3.x in /warehouse/tablespace/external/hive.

The /apps/hive directory, which is the former location of the Hive 2.x warehouse, might or might not exist in HDP 3.x.

For disaster recovery, Hive supports incremental replication of tables from one cluster to another.

ACID table conversion

During the upgrade process, you can override the conversion of ACID v1 tables to ACID v2. For example, you can choose to first convert everything except ACID v1 tables to external tables, and then later convert them to ACID tables one by one.

After upgrading, to convert a non-transactional table to an ACID v2 transactional table, you use the ALTER TABLE command and set table properties to 'transaction'='true'.For example:

```
ALTER TABLE T3 SET TBLPROPERTIES ('transactional'='true');
```

Related Information

Prepare Hive for Upgrade

Prepare Hive for Upgrade: IBM Power Systems

Convert Hive CLI scripts to Beeline

If you have legacy scripts that execute Hive queries from edge nodes using the Hive CLI, you must solve potential incompatibilities with variable substitution in these scripts. HDP 3.0 and later supports Beeline instead of Hive CLI. You can use Beeline to run legacy scripts with a few caveats.

About this task

In this task, you resolve incompatibilities in legacy Hive CLI scripts and Beeline:

- Configuration variables
  - Problem: You cannot refer to configuration parameters in scripts using the hiveconf namespace unless allowed.
  - Solution: You include the parameter in the HiveServer whitelist.
- Namespace problems
  - Problem: Beeline does not support the system and env namespaces for variables.
  - Solution: You remove these namespace references from scripts using a conversion technique described in this task.

Procedure

1. Create a conversion script named env_to_hivevar.sh that removes env references in your SQL scripts.

```
#!/usr/bin/env bash

CMD_LINE=""

#Blank conversion of all env scoped values
for I in `env`; do
  CMD_LINE="$CMD_LINE --hivevar env:${I} "
done
echo ${CMD_LINE}
```

2. On the command line of a node in your cluster, define and export a variable named HIVEVAR, for example, and set it to execute the conversion script.

```
export HIVEVAR=`./env_to_hivevar.sh`
```

3. Define and export variables to hold a few variables for testing the conversion.

```
export LOC_TIME_ZONE="US/EASTERN"
export MY_TEST_VAR="TODAY"
```

4. On the command line of a cluster node, test the conversion: Execute a command that references HIVEVAR to parse a SQL statement, remove the incompatible env namespace, and execute the remaining SQL.

```
hive ${HIVEVAR} -e 'select "${env:LOC_TIME_ZONE}";'
```

```
+-------------+
|     _c0     |
+-------------+
| US/EASTERN  |
+-------------+
```

5. Create a text file named init_var.sql to simulate a legacy script that sets two configuration parameters, one in the problematic env namespace.

```
set mylocal.test.var=hello;
set mylocal.test.env.var=${env:MY_TEST_VAR};
```

6. Whitelist these configuration parameters: In Ambari, go to Hive > Configs > Advanced > Custom hiveserver2-site.
7. Add the property key: hive.security.authorization.sqlstd.confwhitelist.append.
8. Provide the property value, or values, to whitelist, for example: mylocal\..*|junk.

   This action appends mylocal.test.var and mylocal.test.env.var parameters to the whitelist.
9. Save configuration changes, and restart any components as required.
10. Execute a command that references HIVEVAR to parse a SQL script, removes the incompatible env namespace, and executes the remaining SQL, including the whitelisted configuration parameters identified by hiveconf:.

```
hive -i init_var.sql ${HIVEVAR} -e 'select
 "${hiveconf:mylocal.test.var}","${hiveconf:mylocal.test.env.var}";'
```

```
+--------+--------+
|  _c0   |  _c1   |
+--------+--------+
| hello  | TODAY  |
+--------+--------+
```

Related Information

Apache Wiki: Language Manual Variable Substitution

Hive Semantic and Syntax Changes

Creating a table

To improve useability and functionality, Hive 3 significantly changed table creation.

Hive has changed table creation in the following ways:

- Creates ACID-compliant table, which is the default in HDP
- Supports simple writes and inserts
- Writes to multiple partitions
- Inserts multiple data updates in a single SELECT statement
- Eliminates the need for bucketing.

If you have an ETL pipeline that creates tables in Hive, the tables will be created as ACID. Hive now tightly controls access and performs compaction periodically on the tables. The way you access managed Hive tables from Spark and other clients changes. In CDP, access to external tables requires you to set up security access permissions.

Before Upgrade

In HDP 2.6.5, by default CREATE TABLE created a non-ACID table.

After Upgrade

By default CREATE TABLE creates a full, ACID transactional table in ORC format.

Action Required

To access Hive ACID tables from Spark, you connect to Hive using the Hive Warehouse Connector (HWC). To write ACID tables to Hive from Spark, you use the HWC and HWC API. Set up Ranger policies and HDFS ACLs for tables.

Escaping db.table references

You need to change queries that use db.table references to prevent Hive from interpreting the entire db.table string as the table name.

About this task

For ANSI SQL compliance, Hive 3.x rejects `db.table` in SQL queries. A dot (.) is not allowed in table names. You enclose the database name and the table name in backticks.

Procedure

1. Find a table having the problematic table reference.

```
math.students
```

appears in a CREATE TABLE statement.

2. Enclose the database name and the table name in backticks.

```
CREATE TABLE `math`.`students` (name VARCHAR(64), age INT, gpa
DECIMAL(3,2));
```

Casting timestamps

Results of applications that cast numerics to timestamps differ from Hive 2 to Hive 3. Apache Hive changed the behavior of CAST to comply with the SQL Standard, which does not associate a time zone with the TIMESTAMP type.

Before Upgrade

Casting a numeric type value into a timestamp could be used to produce a result that reflected the time zone of the cluster. For example, 1597217764557 is 2020-08-12 00:36:04 PDT. Running the following query casts the numeric to a timestamp in PDT:

```
> SELECT CAST(1597217764557 AS TIMESTAMP);
| 2020-08-12 00:36:04 |
```

After Upgrade

Casting a numeric type value into a timestamp produces a result that reflects the UTC instead of the time zone of the cluster. Running the following query casts the numeric to a timestamp in UTC.

```
> SELECT CAST(1597217764557 AS TIMESTAMP);
| 2020-08-12 07:36:04.557  |
```

Action Required

Change applications. Do not cast from a numeral to obtain a local time zone. Built-in functions from_utc_timestamp and to_utc_timestamp can be used to mimic behavior before the upgrade.

Renaming tables

To harden the system, Hive data can be stored in HDFS encryption zones. RENAME has been changed to prevent moving a table outside the same encryption zone or into a no-encryption zone.

Before Upgrade

Renaming a managed table moves its HDFS location.

After Upgrade

Renaming a managed table moves its location only if the table is created without a LOCATION clause and is under its database directory.

Action Required

None

Checking compatibility of column changes

A default configuration change can cause applications that change column types to fail.

Before Upgrade

In HDP 2.x hive.metastore.disallow.incompatible.col.type.changes is false by default to allow changes to incompatible column types. For example, you can change a STRING column to a column of an incompatible type, such as MAP<STRING, STRING>. No error occurs.

After Upgrade

The hive.metastore.disallow.incompatible.col.type.changes is true by default. Hive prevents changes to incompatible column types. Compatible column type changes, such as INT, STRING, BIGINT, are not blocked.

Action Required

Change applications to disallow incompatible column type changes to prevent possible data corruption.

Dropping partitions

The OFFLINE and NO_DROP keywords in the CASCADE clause for dropping partitions causes performance problems and is no longer supported.

Before Upgrade

You could use OFFLINE and NO_DROP keywords in the CASCADE clause to prevent partitions from being read or dropped.

After Upgrade

OFFLINE and NO_DROP are not supported in the CASCADE clause.

Action Required

Change applications to remove OFFLINE and NO_DROP from the CASCADE clause. Use an authorization scheme, such as Ranger, to prevent partitions from being dropped or read.

Install the Hive service

In Ambari, you install the Hive service as you would any other service, and in the process, you configure Hive to use a database for the Hive Metastore. You can use Oracle, PostgreSQL, or MySQL as the backend database.

About this task

If you use PostgreSQL as the backend database, a supported version later than 9.6 is recommended. Using PostgreSQL 9.6 or earlier can cause problems. Hive Metastore uses hash indexes for PostgreSQL. Hive also uses hash indexes for ACID transactions. TC_TXNID_INDEX and HL_TXNID_INDEX can become corrupted. The workaround is to reindex the corrupted indexes. For example, in PostgreSQL run reindex index tc_txnid_index.

Procedure

1. Configure a database for the Hive Metastore.

- Choose to use an existing database suitable for production work and perform the next step.

- Choose an Ambari installed database, such as MySQL, suitable for development work only, and skip the next step.

2. If you chose an existing database in the last step, create a hive user and database.

   If Ambari installs the database, you do not need to create a hive user and database.

   For example, create user named hive and a database named hive, using the following command, but substituting a password of your choosing for [HIVE_PASSWORD] and the Hive MetaStore FQDN for [HIVE_METASTORE_FQDN]:

```
# mysql -u root -p
CREATE USER 'hive'@'localhost' IDENTIFIED BY '[HIVE_PASSWORD]';
GRANT ALL PRIVILEGES ON *.* TO 'hive'@'localhost';

CREATE USER 'hive'@'%' IDENTIFIED BY '[HIVE_PASSWORD]';
GRANT ALL PRIVILEGES ON *.* TO 'hive'@'%';

CREATE USER 'hive'@'[HIVE_METASTORE_FQDN]' IDENTIFIED BY
'[HIVE_PASSWORD]';
GRANT ALL PRIVILEGES ON *.* TO 'hive'@'[HIVE_METASTORE_FQDN]';

FLUSH PRIVILEGES;

CREATE DATABASE hive;
```

3. Obtain the JDBC driver for the database to the Ambari server node, and as root, set up the driver:

```
ambari-server setup --jdbc-db=<database name> --jdbc-driver=<path to
 driver>
```

4. In Ambari Services > Hive > Configs, test the connection to the database.
5. Follow prompts from the Ambari wizard to complete the installation.

Apache Hive content roadmap

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

Table 2: Apache 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.
Installing and Upgrading	Ambari Install Guide	Hortonworks	Describes Ambari, 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.
	Ambari Upgrade Guide	Hortonworks	Covers how Ambari and the HDP Stack being managed by Ambari can be upgraded independently, getting ready to upgrade Ambari and HDP, upgrading Ambari, and upgrading HDP.
	Installing Hive	Apache wiki	Describes how to install Apache Hive separate from the HDP environment.

Task	Resources	Source	Description
	Configuring Hive	Apache wiki	Describes how to configure Apache Hive separate from the HDP environment and troubleshoot Hive in HDP.
Administering	Setting Up the Metastore	Apache wiki	Describes the metastore parameters.
	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	Materialized Views	Apache wiki	Covers accelerating query processing in data warehouses by pre-computing summaries using materialized views.
	Hive transactions	Apache wiki	Describes ACID operations in Hive.
	Hive Streaming API	Apache wiki	Explains how to use an API for pumping data continuously into Hive using clients such as NiFi and Flume.
	Hive Operators and Functions	Apache wiki	Describes the Language Manual UDF.
	Beeline: HiveServer2 Client	Apache wiki	Describes how to use the Beeline client.
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), which makes Hive faster by using persistent query infrastructure and optimized data caching.
Hive-Spark Integration	Integrating Apache Hive with Apache Spark - Hive Warehouse Connector	Hortonworks Community Connection	Describes how to read and write data between Spark and Hive.
Hive-HBase Integration	HBaseIntegration wiki	Apache wiki	Describes how to integrate the two data access components so that Hive 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	
	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	