

Migrating Hive Workloads to CDP Private Cloud

Date published: 2019-08-22

Date modified: 2023-02-03



Legal Notice

© Cloudera Inc. 2025. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

Migrating Hive workloads from CDH.....	6
Changes to CDH Hive Tables.....	6
Configuration changes.....	7
Hive Configuration Property Changes.....	7
Customizing critical Hive configurations.....	15
Setting Hive Configuration Overrides.....	16
Hive Configuration Requirements and Recommendations.....	17
Configuring HMS for high availability.....	19
Setting up Hive metastore for Atlas.....	20
Changing the Hive warehouse location.....	21
Security tasks.....	21
Making the Hive plugin for Ranger visible.....	21
Configuring authorization to tables.....	23
Setting up access control lists.....	23
Configure encryption zone security.....	24
Configure edge nodes as gateways.....	24
Configure HiveServer HTTP mode.....	24
Key syntax changes.....	25
Handling table reference syntax.....	25
LOCATION and MANAGEDLOCATION clauses.....	25
Key semantic changes and workarounds.....	26
Changing incompatible column types.....	26
Understanding CREATE TABLE behavior.....	27
Configuring legacy CREATE TABLE behavior.....	28
Dropping partitions.....	29
Handling the Keyword APPLICATION.....	29
Handling output of greatest and least functions.....	30
Renaming tables.....	30
TRUNCATE TABLE on an external table.....	30
Other syntax and semantic changes.....	31
Syntax and semantic changes CDH 6.2.1 to CDP 7.0.3.2.....	31
Semantic changes and workarounds CDP 7.1.1.....	41
Semantic changes and workarounds CDP 7.1.4.....	42
Semantic changes and workarounds CDP 7.1.5.....	43
Semantic changes and workarounds CDP 7.1.6.....	44
Semantic changes and workarounds CDP 7.1.7.....	46
Semantic changes and workarounds CDP 7.1.7 SP1.....	46
Semantic changes and workarounds CDP 7.1.7 SP2.....	47
Semantic changes and workarounds CDP 7.1.7 SP2 CHFx.....	48
Semantic changes and workarounds CDP 7.1.8 CHFx.....	49
Migrating Spark Apps.....	50
Preventing SparkSQL incompatibility.....	50
Spark integration with Hive.....	51
Removing Hive on Spark Configurations.....	51
Disabling Partition Type Checking.....	51
Converting Hive CLI scripts to Beeline.....	52
Hive unsupported interfaces and features.....	53

Migrating Hive Workloads from HDP 2.6.5 after an in-place upgrade..... 55

Changes to HDP Hive tables.....	55
Checking and correcting Hive table locations.....	56
Configuration changes.....	57
Hive Configuration Property Changes.....	57
Customizing critical Hive configurations.....	66
Setting Hive Configuration Overrides.....	66
Hive Configuration Requirements and Recommendations.....	67
Configuring HMS for high availability.....	69
Setting up Hive metastore for Atlas.....	70
Changing the Hive warehouse location.....	71
Removing the LLAP Queue.....	72
Security tasks.....	72
Making the Hive plugin for Ranger visible.....	72
Configuring authorization to tables.....	73
Setting up access control lists.....	74
Configure encryption zone security.....	74
Configure edge nodes as gateways.....	75
Configure HiveServer HTTP mode.....	75
Handling syntax changes.....	75
Handling table reference syntax.....	76
LOCATION and MANAGEDLOCATION clauses.....	76
Key semantic changes and workarounds.....	77
Casting timestamps.....	77
Changing incompatible column types.....	77
Understanding CREATE TABLE behavior.....	78
Configuring legacy CREATE TABLE behavior.....	79
Dropping partitions.....	80
Handling output of greatest and least functions.....	80
Renaming tables.....	81
TRUNCATE TABLE on an external table.....	81
Migrating Spark Apps.....	81
Spark integration with Hive.....	81
Identifying and fixing invalid Hive schema versions.....	82
Fixing statistics.....	82
Converting Hive CLI scripts to Beeline.....	83
Hive unsupported interfaces and features.....	84

Replicating Hive data from HDP 3 to CDP.....85

Replicating Hive data.....	86
Configuring the CDP cluster.....	86
Mandatory CDP policy-level properties.....	86
Optional CDP policy-level properties.....	87
Supported scheduled query operations.....	88
Configuring the HDP cluster.....	89
Mandatory HDP cluster configuration properties.....	90
Mandatory HDP policy-level properties.....	91
Optional HDP policy-level properties.....	91
Configuring wire-encrypted clusters.....	92
Example commands for replicating HDP 3 workloads.....	93
Troubleshooting Hive replication using REPL.....	95
Repl Command Known Issues.....	95
Patches Required on HDP.....	96

Patches required on CDP.....	97
Verifying the Hive data replication.....	97
Setting up the HDP cluster.....	99
Verifying replication.....	101
Handling a failed verification.....	102
Validating external table replication.....	103
Enabling background threads after migration.....	104
Migration paths from HDP 3 to CDP for LLAP users.....	105
Migration paths for Hive users.....	105
Migration to Cloudera Private Cloud Base or CDP Public Cloud.....	106
Migration to Cloudera Data Warehouse.....	106
Apache Tez processing of Hive jobs.....	107
Migration paths for Spark users.....	107
Migration to Cloudera Private Cloud Base.....	108
HWC changes from HDP to CDP.....	109
Migrating Hive workloads from Cloudera Base on premises to Cloudera	
Data Warehouse on premises.....	110
Planning a Cloudera Data Warehouse Virtual Warehouse instance.....	111
Apache Tez processing of Hive jobs.....	112
Migrate Hive workloads from HDP (LLAP) to Cloudera Data Warehouse (LLAP).....	113
Migrate from Cloudera Base on premises (Hive on Tez) to Cloudera Data Warehouse (LLAP).....	115
Migrating Hive workloads to ACID.....	116
Tables in Hive 1 and 2 vs. Hive 3.....	118
Compatible storage formats.....	118
Table design considerations.....	118
Hive ingest patterns introduction.....	119
Classic ingest patterns.....	120
ACID ingest patterns.....	123
Handling government regulations in ACID tables.....	130
Key concepts about ACID ingest patterns.....	130

Migrating Hive workloads from CDH

You upgraded from CDH 5.13 - 5.16, or CDH 6, to Cloudera Base on premises. The upgrade moved the Hive data and schema to Cloudera Base on premises. As the Hive Administrator, you need to make Hive tables available to your users. You need to configure your Hive-related services for Cloudera, and secure access to Hive data.

Assumptions

- You are familiar with Apache Hive 3.1 key features and supported interfaces.
- You acquired basic information about the Cloudera platform before you upgraded from CDH.

The configuration changes you need to make are described in the subtopics "Configuration changes". The security tasks you need to perform to secure access to Hive data are described in the subtopics "Security tasks".

Related Information

[Apache Hive 3 Key Features](#)

[Apache Hive 3 Architectural Overview](#)

Changes to CDH Hive Tables

As a Data Scientist, Architect, Analyst, or other Hive user you need to locate and use your Apache Hive 3 tables after an upgrade. You also need to understand the changes that occur during the upgrade process. The location of existing tables after a CDH to Cloudera upgrade does not change. Upgrading CDH to CDP Private Cloud Base converts Hive managed tables to external tables in Hive 3.

About this task

When the upgrade process converts a managed table to external, it sets the table property `external.table.purge` to true. The table is equivalent to a managed table having `purge` set to true in your old CDH cluster.

Managed tables on the HDFS in `/user/hive/warehouse` before the upgrade remain there after the conversion to external. Tables that were external before the upgrade are not relocated. You need to set HDFS policies to access external tables in Ranger, or set up HDFS ACLs.

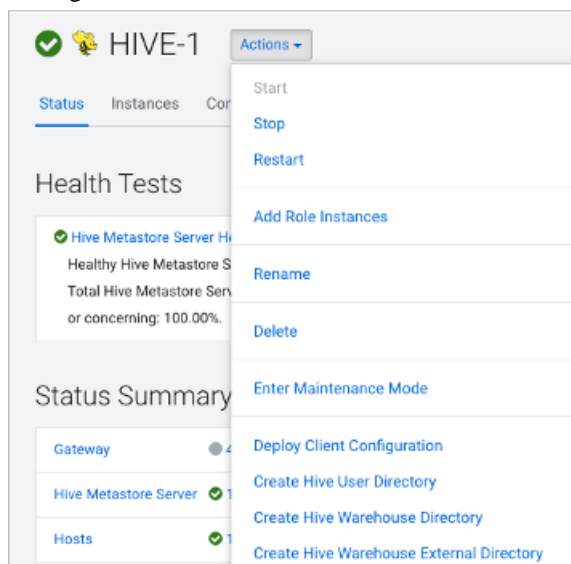
The upgrade process sets the `hive.metastore.warehouse.dir` property to `/warehouse/tablespace/managed/hive`, designating it the Hive warehouse location for managed tables. New managed tables that you create in Cloudera are stored in the Hive warehouse. New external tables are stored in the Hive external warehouse `/warehouse/tablespace/external/hive`.

To change the location of the Hive warehouses, you navigate to one of the following menu items in the first step below.

- Hive Action Menu Create Hive Warehouse Directory
- Hive Action Menu Create Hive Warehouse External Directory

Procedure

1. Set up directories for the Hive warehouse directory and Hive warehouse external directory from Cloudera Manager Actions.



2. In Cloudera Manager, click Clusters Hive (the Hive Metastore service) Configuration , and change the hive.metastore.warehouse.dir property value to the path you specified for the new Hive warehouse directory.
3. Change the hive.metastore.warehouse.external.dir property value to the path you specified for the Hive warehouse external directory.
4. Configure Ranger policies or set up ACL permissions to access the directories.

Related Information

[HDFS ACLS](#)

Configuration changes

Hive Configuration Property Changes

You need to know the property value changes made by the upgrade process as the change might impact your work. You might need to consider reconfiguring property value defaults that the upgrade changes.

Hive Configuration Property Values

The upgrade process changes the default values of some Hive configuration properties and adds new properties. The following list describes those changes that occur after upgrading from CDH or HDP to Cloudera.

datanucleus.connectionPool.maxPoolSize

Before upgrade: 30

After upgrade: 10

datanucleus.connectionPoolingType

Before upgrade: BONECP

After upgrade: HikariCP

hive.auto.convert.join.noconditionaltask.size

Before upgrade: 20971520

After upgrade: 52428800

Exception: Preserves pre-upgrade value if old default is overridden; otherwise, uses new default.

hive.auto.convert.sortmerge.join

Before upgrade: FALSE in the old CDH; TRUE in the old HDP.

After upgrade: TRUE

hive.auto.convert.sortmerge.join.to.mapjoin

Before upgrade: FALSE

After upgrade: TRUE

hive.cbo.enable

Before upgrade: FALSE

After upgrade: TRUE

hive.cbo.show.warnings

Before upgrade: FALSE

After upgrade: TRUE

hive.compactor.worker.threads

Before upgrade: 0

After upgrade: 5

hive.compute.query.using.stats

Before upgrade: FALSE

After upgrade: TRUE

hive.conf.hidden.list

Before upgrade:

```
javax.jdo.option.ConnectionPassword,hive.server2.keystore.password,hive.metastore.dbaccess.ssl.truststore.password,fs.s3.awsAccessKeyId,fs.s3.awsSecretAccessKey,fs.s3n.awsAccessKeyId,fs.s3n.awsSecretAccessKey,fs.s3a.access.key,fs.s3a.secret.key,fs.s3a.proxy.password,dfs.adls.oauth2.credential,fs.adl.oauth2.credential,fs.azure.account.oauth2.client.secret
```

After upgrade:

```
javax.jdo.option.ConnectionPassword,hive.server2.keystore.password,hive.druid.metadata.password,hive.driver.parallel.compilation.global.limit
```

hive.conf.restricted.list

Before upgrade:

```
hive.security.authenticator.manager,hive.security.authorization.manager,hive.users.in.admin.role,hive.server2.xsrf.filter.enabled,hive.spark.client.connect.timeout,hive.spark.client.server.connect.timeout,hive.spark.client.channel.log.level,hive.spark.client.rpc.max.size,hive.spark.client.rpc.threads,hive.spark.client.secret.bits,hive.spark.client.rpc.server.address,hive.spark.client.rpc.server.port,hive.spark.client.rpc.sasl.mechanisms,hadoop.bin.path,yarn.bin.path,spark.home,bonecp.,hikaricp.,hive.driver.parallel.compilation.global.limit,_hive.local.session.path,_hive.hdfs.session.path,_hive.tmp_table_space,_hive.local.session.path,_hive.hdfs.session.path,_hive.tmp_table_space
```


After upgrade:

```
hive.security.authenticator.manager,hive.security.authorization.
manager,hive.security.metastore.authorization.manager,hive.secur
ity.metastore.authenticator.manager,hive.users.in.admin.role,hiv
e.server2.xsrf.filter.enabled,hive.security.authorization.enable
d,hive.distcp.privileged.doAs,hive.server2.authentication.ldap.b
aseDN,hive.server2.authentication.ldap.url,hive.server2.authenti
cation.ldap.Domain,hive.server2.authentication.ldap.groupDNPatte
rn,hive.server2.authentication.ldap.groupFilter,hive.server2.aut
hentication.ldap.userDNPattern,hive.server2.authentication.ldap.
userFilter,hive.server2.authentication.ldap.groupMembershipKey,h
ive.server2.authentication.ldap.userMembershipKey,hive.server2.a
uthentication.ldap.groupClassKey,hive.server2.authentication.lda
p.customLDAPQuery,hive.privilege.synchronizer.interval,hive.spar
k.client.connect.timeout,hive.spark.client.server.connect.timeou
t,hive.spark.client.channel.log.level,hive.spark.client.rpc.max.
size,hive.spark.client.rpc.threads,hive.spark.client.secret.bits
,hive.spark.client.rpc.server.address,hive.spark.client.rpc.serv
er.port,hive.spark.client.rpc.sasl.mechanisms,bonecp.,hive.druid
.broker.address.default,hive.druid.coordinator.address.default,h
ikaricp.,hadoop.bin.path,yarn.bin.path,spark.home,hive.driver.pa
rallel.compilation.global.limit,_hive.local.session.path,_hive.h
dfs.session.path,_hive.tmp_table_space,_hive.local.session.path,
_hive.hdfs.session.path,_hive.tmp_table_space
```

hive.default.fileformat.managed

Before upgrade: None

After upgrade: ORC

hive.default.rcfile.serde

Before upgrade: org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe

After upgrade: org.apache.hadoop.hive.serde2.columnar.LazyBinaryColumnarSerDe

Not supported in Impala. Impala cannot read Hive-created RC tables.

hive.driver.parallel.compilation

Before upgrade: FALSE

After upgrade: TRUE

hive.exec.dynamic.partition.mode

Before upgrade: strict

After upgrade: nonstrict

In Cloudera Base on premises, accidental use of dynamic partitioning feature is not prevented by default.

hive.exec.max.dynamic.partitions

Before upgrade: 1000

After upgrade: 5000

In Cloudera Base on premises, fewer restrictions on dynamic partitioning occur than in the pre-upgrade CDH or HDP cluster.

hive.exec.max.dynamic.partitions.pernode

Before upgrade: 100

After upgrade: 2000

In Cloudera Base on premises, fewer restrictions on dynamic partitioning occur than in the pre-upgrade CDH or HDP cluster.

hive.exec.post.hooks

Before upgrade:

```
com.cloudera.navigator.audit.hive.HiveExecHookContext,org.apache.hadoop.hive ql.hooks.LineageLogger
```

After upgrade: org.apache.hadoop.hive.ql.hooks.HiveProtoLoggingHook

A prime number is recommended.

hive.exec.reducers.max

Before upgrade: 1099

After upgrade: 1009

Exception: Preserves pre-upgrade value if old default is overridden; otherwise, uses new default

hive.execution.engine

Before upgrade: mr

After upgrade: tez

Tez is now the only supported execution engine, existing queries that change execution mode to Spark or MapReduce within a session, for example, fail.

hive.fetch.task.conversion

Before upgrade: minimal

After upgrade: more

hive.fetch.task.conversion.threshold

Before upgrade: 256MB

After upgrade: 1GB

Exception: Preserves pre-upgrade value if old default is overridden; otherwise, uses new default.

hive.hashtable.key.count.adjustment

Before upgrade: 1

After upgrade: 0.99

Exception: Preserves pre-upgrade value if old default is overridden; otherwise, uses new default.

hive.limit.optimize.enable

Before upgrade: FALSE

After upgrade: TRUE

hive.limit.pushdown.memory.usage

Before upgrade: 0.1

After upgrade: 0.04

Exception: Preserves pre-upgrade value if old default is overridden; otherwise, uses new default.

hive.mapjoin.hybridgrace.hashtable

Before upgrade: TRUE

After upgrade: FALSE

hive.mapred.reduce.tasks.speculative.execution

Before upgrade: TRUE

After upgrade: FALSE

hive.metastore.aggregate.stats.cache.enabled

Before upgrade: TRUE

After upgrade: FALSE

hive.metastore.disallow.incompatible.col.type.changes

Before upgrade: FALSE

After upgrade: TRUE

Schema evolution is more restrictive in Cloudera Base on premises than in CDH to avoid data corruption. The new default disallows column type changes if the old and new types are incompatible.

hive.metastore.dml.events

Before upgrade: FALSE

After upgrade: TRUE

hive.metastore.event.message.factory

Before upgrade: org.apache.hadoop.hive.metastore.messaging.json.ExtendedJSONMessageFactory

After upgrade: org.apache.hadoop.hive.metastore.messaging.json.gzip.GzipJSONMessageEncoder

hive.metastore.uri.selection

Before upgrade: SEQUENTIAL

After upgrade: RANDOM

hive.metastore.warehouse.dir

Before upgrade from CDH: /user/hive/warehouse

Before upgrade from HDP: /apps/hive/warehouse

After upgrade from CDH: /warehouse/tablespace/managed/hive

After upgrade from HDP: /warehouse/tablespace/managed/hive

For information about the location of old tables and new tables, which you create after the upgrade, see [Changes to CDH Hive Tables](#) or [Changes to HDP Hive tables](#).

hive.optimize.metadatanonly

Before upgrade: FALSE

After upgrade: TRUE

hive.optimize.point.lookup.min

Before upgrade: 31

After upgrade: 2

hive.prewarm.numcontainers

Before upgrade: 10

After upgrade: 3

hive.script.operator.env.blacklist

Before upgrade: hive.txn.valid.txns,hive.script.operator.env.blacklist

After upgrade: hive.txn.valid.txns,hive.txn.tables.valid.writeids,hive.txn.valid.writeids,hive.script.operator.env.blacklist

hive.security.authorization.sqlstd.confwhitelist

Before upgrade:

```
hive\auto\.*hive\cbo\.*hive\convert\.*hive\exec\dynamic\
.partition.*hive\exec\.*\dynamic\partitions\.*hive\exec\c
ompress\.*hive\exec\infer\.*hive\exec\mode.local\.*hive\
exec\orc\.*hive\exec\parallel.*hive\explain\.*hive\fetch.
task\.*hive\groupby\.*hive\hbase\.*hive\index\.*hive\ind
ex\.*hive\intermediate\.*hive\join\.*hive\limit\.*hive\l
og\.*hive\mapjoin\.*hive\merge\.*hive\optimize\.*hive\or
c\.*hive\outerjoin\.*hive\parquet\.*hive\ppd\.*hive\prew
arm\.*hive\server2\proxy\userhive\skewjoin\.*hive\smbjoin
\.*hive\stats\.*hive\strict\.*hive\tez\.*hive\vectorized
\.*mapred\map\.*mapred\reduce\.*mapred\output\compression
\codecmapped\job\queuenamemapred\output\compression\typema
pred\min\split\sizemapreduce\job\reduce\slowstart\complet
edmapsmappedreduce\job\queuenamemapreduce\job\tagmapreduce\in
put\fileinputformat\split\minsizemapreduce\map\.*mapreduce\
.reduce\.*mapreduce\output\fileoutputformat\compress\codecm
apreduce\output\fileoutputformat\compress\typeoozie\.*tez\
am\.*tez.task\.*tez.runtime\.*tez.queue\namehive\transpo
se\aggr\joinhive\exec\reducers\bytes\per.reducerhive\cli
ent\stats\countershive\exec\default\partition\namehive\ex
ec\drop\ignorenonexistenthive\counters\group\namehive\defa
ult\fileformat\managedhive\enforce\bucketmapjoinhive\enforc
e\sortmergebucketmapjoinhive\cache\expr\evaluationhive\quer
y\result\fileformathive\hashtable\loadfactorhive\hashtable\
.initialCapacityhive\ignore\mapjoin\hinthive\limit\row\max
\sizehive\mapred\modehive\map\aggrhive\compute\query\usi
ng\statshive\exec\rowoffsethive\variable\substitutehive\va
riable\substitute\depthhive\autogen\columnalias\prefix\inc
ludefuncnamehive\autogen\columnalias\prefix\labelhive\exec\
.check\crossproductshive\cli\tez\session\asynhive\compath
ive\exec\concatenate\check\indexhive\display\partition\co
ls\separatelyhive\error\on\empty\partitionhive\execution\
enginehive\exec\copyfile\maxsizehive\exim\uri\scheme\whit
elisthive\file\max\footerhive\insert\into\multilevel\dirs
hive\localize\resource\num\wait\attemptshive\multi\insert
\move\tasks\share\dependencieshive\support\quoted\identif
iershive\resultset\use\unique\column\namehive\analyze\st
mt\collect\partlevel\statshive\exec\schema\evolutionhive\
server2\logging\operation\levelhive\server2\thrift\results
et\serialize\in\taskshive\support\special\characters\tabl
enamehive\exec\job\debug\capture\stacktraceshive\exec\job
\debug\timeouthive\llap\io\enabledhive\llap\io\use\file
id\pathhive\llap\daemon\service\hostshive\llap\execution\
.modehive\llap\auto\allow\uberhive\llap\auto\enforce\tre
ehive\llap\auto\enforce\vectorizedhive\llap\auto\enforce\
.statshive\llap\auto\max\input\sizehive\llap\auto\max\o
utput\sizehive\llap\skip\compile\udf\checkhive\llap\clie
nt\consistent\splitshive\llap\enable\grace\join\in\llaph
ive\llap\allow\permanent\fnshive\exec\max\created\filesh
ive\exec\reducers\maxhive\reorder\nway\joinshive\output\
file\extensionhive\exec\show\job\failure\debug\infohive\
exec\tasklog\debug\timeouthive\query\id
```

After upgrade:

```
hive\auto\.*hive\cbo\.*hive\convert\.*hive\druid\.*hive\
exec\dynamic\partition.*hive\exec\max\dynamic\partitions.
.*hive\exec\compress\.*hive\exec\infer\.*hive\exec\mode.l
ocal\.*hive\exec\orc\.*hive\exec\parallel.*hive\exec\que
ry\redactor\.*hive\explain\.*hive\fetch.task\.*hive\group
by\.*hive\hbase\.*hive\index\.*hive\index\.*hive\interme
```

```

diate\...*hive\jdbc\...*hive\join\...*hive\limit\...*hive\log\...
*hive\mapjoin\...*hive\merge\...*hive\optimize\...*hive\materializedview\...
*hive\orc\...*hive\outerjoin\...*hive\parquet\...*hive\ppd\...*hive\prewarm\...
*hive\query\redaction\...*hive\server2\thrift\resultset\default\fetch\sizehive\server2\proxy\userhive\skewjoin\...
*hive\smbjoin\...*hive\stats\...*hive\strict\...*hive\tez\...*hive\vectorized\...
*hive\query\reexecution\...*reexec\overlay\...*fs\defaultFSssl\client\truststore\locationdistcp\atomicdistcp\ignore\failuresdistcp\preserve\statusdistcp\preserve\rawattrsdistcp\sync\foldersdistcp\delete\missing\ sourcedistcp\keystore\resourcedistcp\liststatus\threadsdistcp\max\mapsdistcp\copy\strategydistcp\skip\crcdistcp\copy\overwritedistcp\copy\appenddistcp\map\bandwidth\mbdistcp\dynamic\...
*distcp\meta\folderdistcp\copy\listing\classdistcp\filters\classdistcp\options\skipcrccheckdistcp\options\mdistcp\options\numListstatusThreadsdistcp\options\mapredSslConfdistcp\options\bandwidthdistcp\options\overwritedistcp\options\strategydistcp\options\idistcp\options\p.*distcp\options\updatedistcp\options\deletemapred\map\...
mapred\reduce\...mapred\output\compression\codecmapped\job\queue\name\mapred\output\compression\typemapred\min\split\size\mapreduce\job\reduce\slowstart\completed\map\mapreduce\job\queue\name\mapreduce\job\tag\mapreduce\input\file\input\format\split\min\size\mapreduce\map\...
*mapreduce\reduce\...mapred\reduce\output\file\output\format\compress\codecmapped\output\file\output\format\compress\type\oozie\...
*tez\am\...
*tez\task\...
*tez\runtime\...
*tez\queue\namehive\transpose\aggr\joinhive\exec\reducers\bytes\per\reducerhive\client\stats\count\erhive\exec\default\partition\namehive\exec\drop\ignore\nonexistenthive\counters\group\namehive\default\file\format\managedhive\enforce\bucket\mapjoinhive\enforce\sort\merge\bucket\mapjoinhive\cache\expr\evaluationhive\query\result\file\format\hive\hashtable\loadfactorhive\hashtable\initialCapacityhive\ignore\mapjoin\hinhive\limit\row\max\sizehive\mapred\modehive\map\aggrhive\compute\query\using\statshive\exec\row\offsethive\variable\substitutehive\variable\substitute\depthhive\autogen\columnalias\prefix\include\funcnamehive\autogen\columnalias\prefix\labelhive\exec\check\crossproductshive\cli\tez\session\asynchive\compathive\display\partition\cols\separatelyhive\error\on\empty\partitionhive\execution\enginehive\exec\copyfile\maxsizehive\exim\uri\scheme\whitelisthive\file\max\footerhive\insert\into\multilevel\dirshive\localize\resource\num\wait\attemptshive\multi\insert\move\tasks\share\dependencieshive\query\results\cache\enabledhive\query\results\cache\wait\for\pending\resultshive\support\quoted\identifiershive\resultset\use\unique\column\namehive\analyze\stmt\collect\partlevel\statshive\exec\schema\evolutionhive\server2\logging\operation\levelhive\server2\thrift\resultset\serialize\in\taskshive\support\special\characters\tablenamehive\exec\job\debug\capture\stacktraceshive\exec\job\debug\timeouthive\llap\io\enabledhive\llap\io\use\fileid\pathhive\llap\daemon\service\hostshive\llap\execution\modehive\llap\auto\allow\uberhive\llap\auto\enforce\treehive\llap\auto\enforce\vectorizedhive\llap\auto\enforce\statshive\llap\auto\max\input\sizehive\llap\auto\max\output\sizehive\llap\skip\compile\udf\checkhive\llap\client\consistent\splitshive\llap\enable\grace\join\in\llaphive\llap\allow\permanent\fnshive\exec\max\created\fileshive\exec\reducers\maxhive\reorder\nway\joinshive\output\file\extensionhive\exec\show\job\failure\debug\infohive\exec\tasklog\debug\timeouthive\query\idhive\query\tag

```

hive.security.command.whitelist

Before upgrade: set,reset,dfs,add,list,delete,reload,compile

After upgrade: set,reset,dfs,add,list,delete,reload,compile,llap

hive.server2.enable.doAs

Before upgrade: TRUE (in case of an insecure cluster only)

After upgrade: FALSE (in all cases)

Affects only insecure clusters by turning off impersonation. Permission issues are expected to arise for affected clusters.

hive.server2.idle.session.timeout

Before upgrade: 12 hours

After upgrade: 24 hours

Exception: Preserves pre-upgrade value if old default is overridden; otherwise, uses new default.

hive.server2.max.start.attempts

Before upgrade: 30

After upgrade: 5

hive.server2.parallel.ops.in.session

Before upgrade: TRUE

After upgrade: FALSE

A Tez limitation requires disabling this property; otherwise, queries submitted concurrently on a single JDBC connection fail or execute slower.

hive.server2.support.dynamic.service.discovery

Before upgrade: FALSE

After upgrade: TRUE

hive.server2.tez.initialize.default.sessions

Before upgrade: FALSE

After upgrade: TRUE

hive.server2.thrift.max.worker.threads

Before upgrade: 100

After upgrade: 500

Exception: Preserves pre-upgrade value if the old default is overridden; otherwise, uses new default.

hive.server2.thrift.resultset.max.fetch.size

Before upgrade: 1000

After upgrade: 10000

hive.service.metrics.file.location

Before upgrade: /var/log/hive/metrics-hiveserver2/metrics.log

After upgrade: /var/log/hive/metrics-hiveserver2-hiveontez/metrics.log

This location change is due to a service name change.

hive.stats.column.autogather

Before upgrade: FALSE

After upgrade: TRUE

hive.stats.deserialization.factor

Before upgrade: 1

After upgrade: 10

hive.support.special.characters.tablename

Before upgrade: FALSE

After upgrade: TRUE

hive.tez.auto.reducer.parallelism

Before upgrade: FALSE

After upgrade: TRUE

hive.tez.bucket.pruning

Before upgrade: FALSE

After upgrade: TRUE

hive.tez.container.size

Before upgrade: -1

After upgrade: 4096

hive.tez.exec.print.summary

Before upgrade: FALSE

After upgrade: TRUE

hive.txn.manager

Before upgrade: org.apache.hadoop.hive ql.lockmgr.DummyTxnManager

After upgrade: org.apache.hadoop.hive ql.lockmgr.DbTxnManager

hive.vectorized.execution.mapjoin.minmax.enabled

Before upgrade: FALSE

After upgrade: TRUE

hive.vectorized.execution.mapjoin.native.fast.hashtable.enabled

Before upgrade: FALSE

After upgrade: TRUE

hive.vectorized.use.row.serde.deserialize

Before upgrade: FALSE

After upgrade: TRUE

Customizing critical Hive configurations

As Administrator, you need property configuration guidelines. You need to know which properties you need to reconfigure after upgrading. You must understand which the upgrade process carries over from the old cluster to the new cluster.

The Cloudera upgrade process tries to preserve your Hive configuration property overrides. These overrides are the custom values you set to configure Hive in the old CDH or HDP cluster. The upgrade process does not preserve all overrides. For example, a custom value you set for `hive.exec.max.dynamic.partitions.pernode` is preserved. In the case of other properties, for example `hive.cbo.enable`, the upgrade ignores any override and just sets the Cloudera-recommended value.

The upgrade process does not preserve overrides to the configuration values of the following properties that you likely need to reconfigure to meet your needs:

- `hive.conf.hidden.list`
- `hive.conf.restricted.list`
- `hive.exec.post.hooks`
- `hive.script.operator.env.blacklist`
- `hive.security.authorization.sqlstd.confwhitelist`
- `hive.security.command.whitelist`

The Apache Hive Wiki describes these properties. The values of these properties are lists.

The upgrade process ignores your old list and sets a new generic list. For example, the `hive.security.command.whitelist` value is a list of security commands you consider trustworthy and want to keep. Any overrides of this list that you set in the old cluster are not preserved. The new default is probably a shorter (more restrictive) list than the original default you were using in the old cluster. You need to customize this Cloudera to meet your needs.

Check and change each property listed above after upgrading as described in the next topic.

Consider reconfiguring more property values than the six listed above. Even if you did not override the default value in the old cluster, the Cloudera default might have changed in a way that impacts your work.

Related Information

[Hive Configuration Property Changes](#)

[Hive Configuration Requirements and Recommendations](#)

[Apache Hive Wiki: Configuration Properties](#)

Setting Hive Configuration Overrides

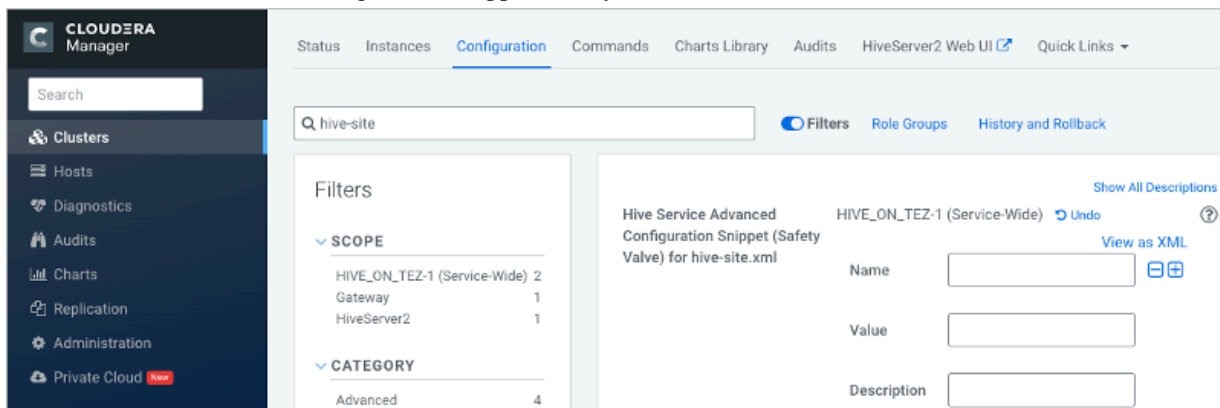
You need to know how to configure the critical customizations that the upgrade process does not preserve from your old Hive cluster. Referring to your records about your old configuration, you follow steps to set at least six critical property values.

About this task

By design, the six critical properties that you need to customize are not visible in Cloudera Manager, as you can see from the Visible in Cloudera Manager column of Configurations Requirements and Recommendations. You use the Safety Valve to add these properties to `hive-site.xml` as shown in this task.

Procedure

1. In Cloudera Manager Clusters select the Hive on Tez service. Click Configuration, and search for `hive-site.xml`.
2. In Hive Service Advanced Configuration Snippet (Safety Valve) for `hive-site.xml`, click +.



3. In Name, add the `hive.conf.hidden.list` property.
4. In Value, add your custom list.
5. Customize the other critical properties: `hive.conf.restricted.list`, `hive.exec.post.hooks`, `hive.script.operator.env.blacklist`, `hive.security.authorization.sqlstd.confwhitelist`, `hive.security.command.whitelist`.
Use `hive.security.authorization.sqlstd.confwhitelist.append`, for example, to set up the list.

6. Save the changes and restart the Hive service.
7. Look at the Configurations Requirements and Recommendations to understand which overrides were preserved or not.

Related Information

[Hive Configuration Property Changes](#)

[Hive Configuration Requirements and Recommendations](#)

[Apache Hive Wiki: Configuration Properties](#)

Hive Configuration Requirements and Recommendations

You need to set certain Hive and HiveServer (HS2) configuration properties after upgrading. You review recommendations for setting up Cloudera Base on premises for your needs, and understand which configurations remain unchanged after upgrading, which impact performance, and default values.

Requirements and Recommendations

The following table includes the Hive service and HiveServer properties that the upgrade process changes. Other property values (not shown) are carried over unchanged from CDH or HDP to Cloudera

- Set After Upgrade column: properties you need to manually configure after the upgrade to Cloudera. Pre-existing customized values are not preserved after the upgrade.
- Default Recommended column: properties that the upgrade process changes to a new value that you are strongly advised to use.
- Impacts Performance column: properties changed by the upgrade process that you set to tune performance.
- Safety Value Overrides column: How the upgrade process handles Safety Valve overrides.
 - Disregards: the upgrade process removes any old CDH Safety Valve configuration snippets from the new CDP configuration.
 - Preserves means the upgrade process carries over any old CDH snippets to the new CDP configuration.
 - Not applicable means the value of the old parameter is preserved.
- Visible in CM column: property is visible in Cloudera Manager after upgrading. Cloudera Manager after upgrading.

If a property is not visible, and you want to configure it, use the Cloudera Manager Safety Valve to safely add the parameter to the correct file, for example to a cluster-wide, hive-site.xml file.

Table 1:

Property	Set After Upgrade	Default Recommended	Impacts Performance	New Feature	Safety Valve Overrides	Visible in CM
datanucleus.connectionPool.maxPoolSize			#		Preserve	
datanucleus.connectionPoolingType			#		Disregard	
hive.async.log.enabled					Disregard	#
hive.auto.convert.join.noconditionaltask.size					Not applicable	#
hive.auto.convert.sortmerge.join					Preserve	
hive.auto.convert.sortmerge.join.to.mapjoin					Preserve	
hive.cbo.enable					Disregard	#
hive.cbo.show.warnings					Disregard	
hive.compactor.worker.threads				#	Disregard	#
hive.compute.query.using.stats			#		Disregard	#
hive.conf.hidden.list	#				Disregard	
hive.conf.restricted.list	#				Disregard	

Property	Set After Upgrade	Default Recommendation	Impacts Performance	New Feature	Safety Valve Overrides	Visible in CM
hive.default.fileformat.managed					Disregard	#
hive.default.rcfile.serde		#			Preserve	
hive.driver.parallel.compilation					Disregard	#
hive.exec.dynamic.partition.mode					Disregard	
hive.exec.max.dynamic.partitions					Preserve	
hive.exec.max.dynamic.partitions.pernode					Preserve	
hive.exec.post.hooks	#				Disregard	
hive.exec.reducers.max		# or other prime number			Not applicable	#
hive.execution.engine					Disregard	
hive.fetch.task.conversion			#		Not applicable	#
hive.fetch.task.conversion.threshold			#		Not applicable	#
hive.hashtable.key.count.adjustment			#		Preserve	
hive.limit.optimize.enable		#			Disregard	
hive.limit.pushdown.memory.usage			#		Not Applicable	#
hive.mapjoin.hybridgrace.hashtable		#	#		Disregard	
hive.mapred.reduce.tasks.speculative.execution		#			Disregard	
hive.metastore.aggregate.stats.cache.enabled		#	#		Disregard	
hive.metastore.disallow.incompatible.col.type.changes					Disregard	
hive.metastore.dml.events					Disregard	#
hive.metastore.event.message.factory		#			Disregard	
hive.metastore.uri.selection		#			Disregard	
hive.metastore.warehouse.dir					Preserve	#
hive.optimize.metadataonly		#			Disregard	
hive.optimize.point.lookup.min					Disregard	
hive.prewarm.numcontainers					Disregard	
hive.script.operator.env.blacklist	#				Disregard	
hive.security.authorization.sqlstd.confwhitelist	#				Disregard	
hive.security.command.whitelist	#				Disregard	
hive.server2.enable.doAs					Disregard	#
hive.server2.idle.session.timeout					Not applicable	#
hive.server2.max.start.attempts					Preserve	
hive.server2.parallel.ops.in.session					Preserve	
hive.server2.support.dynamic.service.discovery				#	Disregard	#
hive.server2.tez.initialize.default.sessions				#	Disregard	
hive.server2.thrift.max.worker.threads					Not Applicable	#
hive.server2.thrift.resultset.max.fetch.size					Preserve	
hive.service.metrics.file.location					Disregard	#

Property	Set After Upgrade	Default Recommendation	Impacts Performance	New Feature	Safety Valve Overrides	Visible in CM
hive.stats.column.autogather		#			Disregard	
hive.stats.deserialization.factor		#			Disregard	
hive.support.special.characters.tablename		#			Disregard	
hive.tez.auto.reducer.parallelism				#	Disregard	#
hive.tez.bucket.pruning				#	Disregard	#
hive.tez.container.size				#	Disregard	#
hive.tez.exec.print.summary				#	Disregard	#
hive.txn.manager				#	Disregard	#
hive.vectorized.execution.mapjoin.minmax.enabled		#			Disregard	
hive.vectorized.execution.mapjoin.native.fast.hashtable.enabled		#			Disregard	
hive.vectorized.use.row.serde.deserialize		#			Disregard	

Configuring HMS for high availability

To provide failover to a secondary Hive metastore if your primary instance goes down, you need to know how to add a Metastore role in Cloudera Manager and configure a property.

About this task

Multiple HMS instances run in active/active mode. No load balancing occurs. An HMS client always reaches the first instance unless it is down. In this case, the client scans the `hive.metastore.uris` property that lists the HMS instances for a replacement HMS. The second HMS is the designated replacement if `hive.metastore.uri.selection` is set to `SEQUENTIAL` (recommended and the default); otherwise, the replacement is selected randomly from the list if `hive.metastore.uri.selection` is set to `RANDOM`.

Before you begin

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

Procedure

1. In Cloudera Manager, click **Clusters** **Hive** **Configuration**.
2. Take one of the following actions:
 - If you have a cluster secured by Kerberos, search for **Hive Delegation Token Store**, which specifies storage for the Kerberos token as described below.
 - If you have an unsecured cluster, skip the next step.
3. Select `org.apache.hadoop.hive.thrift.DBTokenStore`, and save the change.

Show All Descriptions

Hive Metastore Delegation Token Store

hive.cluster.delegation.token.store.class

`hive_metastore_delegation_token_store`

Hive Metastore Server Default Group

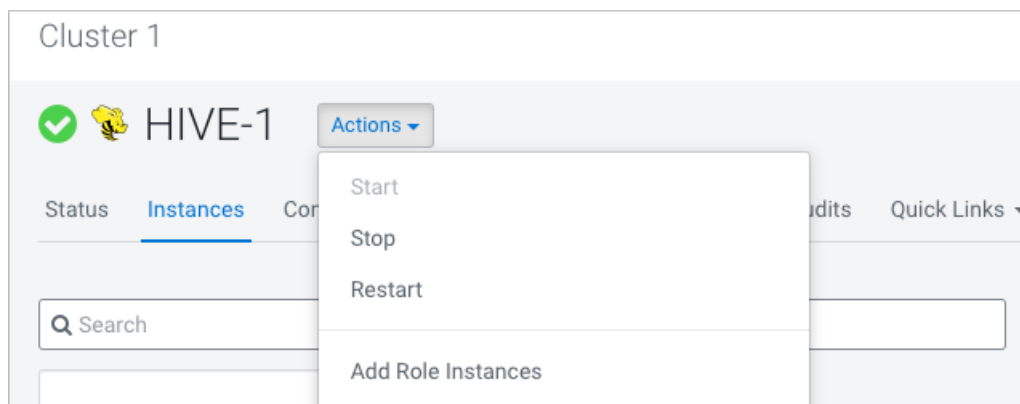
☐ `org.apache.hadoop.hive.thrift.MemoryTokenStore`

☒ `org.apache.hadoop.hive.thrift.DBTokenStore`

☐ `org.apache.hadoop.hive.thrift.ZooKeeperTokenStore`

Storage for the Kerberos delegation token is defined by the `hive.cluster.delegation.token.store.class` property. The available choices are Zookeeper, the Metastore, and memory. Cloudera recommends using the database by setting the `org.apache.hadoop.hive.thrift.DBTokenStore` property.

4. Click Instances Actions Add Role Instances



5. In Assign Roles, in Metastore Server, click Select Hosts.
6. In Hosts Selected, scroll and select the host that you want to serve as the backup Metastore, and click OK.
7. Click Continue until you exit the wizard.
8. Start the Metastore role on the host from the Actions menu.
The hive.metastore.uris property is updated automatically.
9. To check or to change the hive.metastore.uri.selection property, go to Clusters Hive Configurations , and search for Hive Service Advanced Configuration Snippet (Safety Valve) for hive-site.xml.
10. Add the property and value (SEQUENTIAL or RANDOM).

Setting up Hive metastore for Atlas

As Administrator, you might plan to recommend Atlas for Hive metadata management and data governance. You have to check that Hive metastore for Atlas is set up, so users can build catalogs of data assets, classify, and govern the assets. If Atlas is not set up you learn how to do so. This section is not applicable if you are upgrading to CDP Private Cloud Base 7.1.7.

About this task

In this task, you set the name of the Atlas service for Hive metastore to use.

Procedure

1. In Cloudera Manager, click Clusters Hive Configurations .
2. Search for Atlas Service.
3. Choose a method based on the results of your search:
 - If Cloudera Manager finds the Atlas Service, check the checkbox to enable the Hive Metastore hook in your Cloudera Manager instance.
 - If Cloudera Manager does not find the Atlas Service, in Hive Service Advanced Configuration Snippet (Safety Valve) for atlas-application properties, enter an XML snippet in the value element that provides the name of your Atlas service, myatlasservice in the example below.

```
<property>
  <name>atlas_service</name>
  <value>myatlasservice</value>
</property>
```

4. Save changes.
5. Restart the Hive metastore service.

Changing the Hive warehouse location

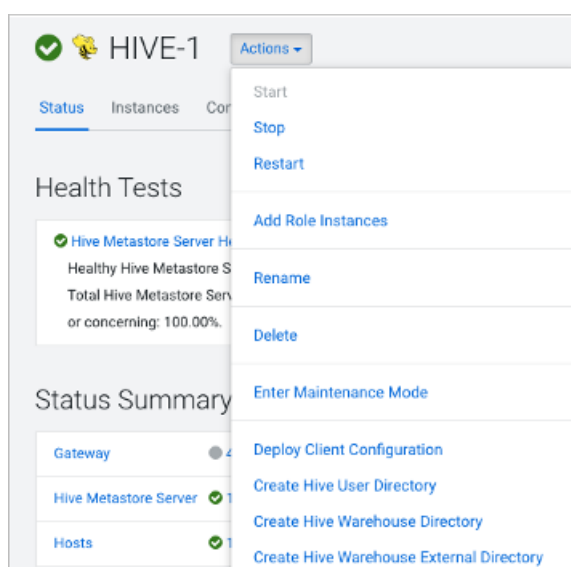
You can change the location of the Hive warehouse by using the configuration settings in your Cloudera Manager instance.

Procedure

1. In Cloudera Manager, click Clusters > Hive (the Hive Metastore service) > Configuration, and change the `hive.metastore.warehouse.dir` property value to the path you specified for the new Hive warehouse directory.
2. Change the `hive.metastore.warehouse.external.dir` property value to the path you specified for the Hive warehouse external directory.
3. Save the above configuration changes.

In Cloudera Manager, navigate to the Hive service and from the Actions drop-down, run the services:

- Create Hive Warehouse Directory
- Create Hive Warehouse External Directory



4. Restart the required services for the changes to take effect.

Related Information

[Ranger RMS Authorization for Hive-HDFS](#)

[HDFS ACL Permissions Model](#)

[HDFS ACLS](#)

Security tasks

After an in-place upgrade to Cloudera, as Administrator, you might need to perform a few security tasks, depending on the type of security you set up, Ranger or HDFS Access Control Lists (ACLs), as well as your data encryption requirements and use of clients to access Hive.

Making the Hive plugin for Ranger visible

After upgrading from HDP or CDH clusters to Cloudera, the Hive plugin for the Hive Metastore and HiveServer2 appears in the Ranger Admin UI unless configuration property problems due to upgrading exist. You can rectify the incorrect properties to fix the problem.

About this task

If the Hive Metastore plugin does not appear in the Ranger Admin UI, you must remove the following property settings from Hive Metastore hive-site.xml safety valve:

- hive.security.authorization.enabled
- hive.security.authorization.manager
- hive.security.metastore.authorization.manager

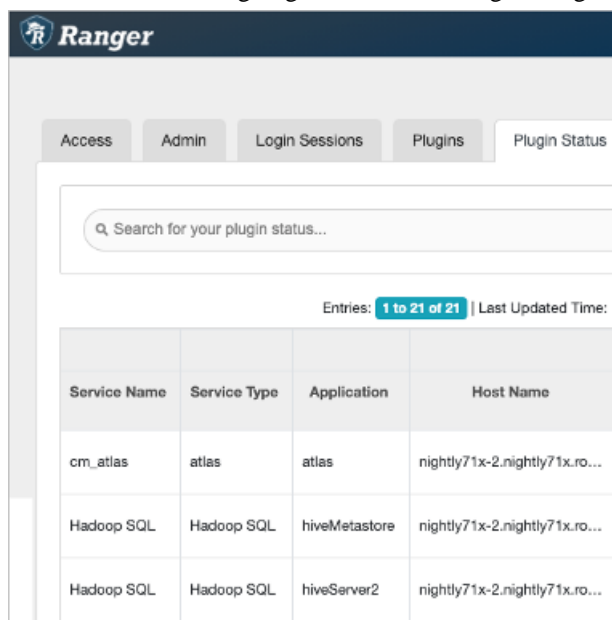
If the HiveServer2 plugin does not appear in the Ranger Admin UI, you must remove the following property settings from HiveServer2 hive-site.xml safety valve:

- hive.security.authorization.enabled
- hive.security.authorization.manager
- hive.security.metastore.authorization.manager
- hive.security.authenticator.manager

After removing these configuration properties, restart the Hive Metastore and HiveServer2 services from Cloudera Manager. Next, you must check whether the Ranger Hive Metastore and HiveServer2 plugins are enabled successfully. To do so:

Procedure

1. From Cloudera Manager, go to Clusters Ranger Ranger Admin Web UI Audit Plugin Status .



Service Name	Service Type	Application	Host Name
cm_atlas	atlas	atlas	nightly71x-2.nightly71x.ro...
Hadoop SQL	Hadoop SQL	hiveMetastore	nightly71x-2.nightly71x.ro...
Hadoop SQL	Hadoop SQL	hiveServer2	nightly71x-2.nightly71x.ro...

The Hadoop SQL service type for the hiveMetastore and hiveServer2 applications should appear. If so, skip the next step. Your configuration is ok.

2. If, after removing the Hive Metastore and HiveServer2 configuration properties from the respective hive-site.xml safety valves, the Hive Metastore and HiveServer2 plugins are NOT visible, you must confirm whether or not the following configuration properties appear in hive-site.xml:

For Hive Metastore, confirm whether or not the following key-value pair appears in hive-site.xml:

Key: hive.metastore.pre.event.listeners

Value: org.apache.hadoop.hive.ql.security.authorization.plugin.metastore.HiveMetaStoreAuthorizer

If this key-value pair does not appear in hive-site.xml, then add it to the Hive Metastore hive-site.xml safety valve.

For HiveServer2, confirm whether or not the following key value pair appears in hive-site.xml:

Key: hive.security.authenticator.manager

Value: org.apache.hadoop.hive.ql.security.SessionStateUserAuthenticator

If this key-value pair does not appear in hive-site.xml, then add it to the HiveServer2 hive-site.xml safety valve.

Configuring authorization to tables

Although the upgrade process makes no change to the location of external tables, you need to set up access to external tables in HDFS. If you choose the recommended Ranger security model for authorization, you need to set up policies and configure Hive metastore (HMS).

About this task

Set up access to external tables in HDFS using one of the following methods.

- Set up a Hive HDFS policy in Ranger (recommended) to include the paths to external table data.
- Put an HDFS ACL in place. Store the external text file, for example a comma-separated values (CSV) file, in HDFS that will serve as the data source for the external table.

If you want to use Ranger to authorize access to your tables, you must configure a few HMS properties for authorization in addition to setting up Ranger policies. If you have not configured HMS, attempting to create a table using Spark SQL, Beeline, or Hue results in the following error:

```
org.apache.hadoop.hive.ql.ddl.DDLTask. MetaException(message:No privilege 'Create' found for outputs { database:DATABASE_NAME, table:TABLE_NAME})
```

Related Information

[Authorizing Apache Hive Access](#)

[Configuring HMS properties for authorization](#)

Setting up access control lists

Several sources of information about setting up HDFS ACLs plus a brief Ranger overview and pointer to Ranger information prepare you to set up Hive authorization.

In Cloudera Base on premises, HDFS supports POSIX ACLs (Access Control Lists) to assign permissions to users and groups. In lieu of Ranger policies, you use HDFS ACLs to check and make any necessary changes in HDFS permission changes. For more information, see HDFS ACLs, Apache Software Foundation HDFS Permissions Guide, and HDFS ACL Permissions.

In Ranger, you give multiple groups and users specific permissions based on your use case. You apply permissions to a directory tree instead of dealing with individual files. For more information, see [Authorizing Apache Hive Access](#).

If possible, you should use Ranger policies over HDFS ACLs to control HDFS access. Controlling HDFS access through Ranger provides a single, unified interface for understanding and managing your overall governance framework and policy design. If you need to mimic the legacy Sentry HDFS ACL Sync behavior for Hive and Impala tables, consider using Ranger RMS.

Related Information

[Ranger RMS Authorization for Hive-HDFS](#)

[HDFS ACLS](#)[Apache Hive 3 Architectural Overview](#)[Configure a Resource-based Policy: Hive](#)[Ranger RMS Authorization for Hive-HDFS](#)[HDFS ACL Permissions Model](#)

Configure encryption zone security

Under certain conditions, you as Administrator, need to perform a security-related task to allow users to access to tables stored in encryption zones. You find out how to prevent access problems to these tables.

About this task

Hive on Tez cannot run some queries on tables stored in encryption zones under certain conditions. Perform the following procedure only when the cluster uses self-signed certificates.



Important: Skip this task for clusters where TLS certificates are properly signed by a Certificate Authority (CA), and the CA is in the truststore files.

Procedure

1. Copy the ssl-client.xml file to a directory that is available on all hosts.
 2. In Cloudera Manager, click Clusters Hive on Tez Configuration .
 3. Search for the Hive Service Advanced Configuration Snippet (Safety Valve) for hive-site.xml setting.
 4. In the Hive Service Advanced Configuration Snippet (Safety Valve) for hive-site.xml setting, click + .
 5. In Name enter the property tez.aux.uris and in value enter path-to-ssl-client.xml.
- Ensure that you include the file URI scheme in the path. For example:

```
tez.aux.uris=file:///etc/hadoop/conf
```

Configure edge nodes as gateways

If you use command-line clients, such as Sqoop, to access Hive, you must configure these gateways to use defaults for your service. You can accomplish this task in a few steps.

About this task

By default, the HS2 instances configured in the migration already have the default beeline-site.xml file defined for the service. Other hosts do not. Configure these hosts as a gateway for that service.

Procedure

1. Find the notes you made before the upgrade about edge nodes and default, connected endpoints.
2. In Cloudera Manager, configure hosts other than HiveServer (HS2) hosts that you want to be Hive Gateway nodes as gateways for the default beeline-site.xml file for the gateway service.

Configure HiveServer HTTP mode

If you use Knox, you might need to change the HTTP mode configuration. If you installed Knox on Cloudera Base on premises and want to proxy HiveServer with Knox, you need to change the default HiveServer transport mode (hive.server2.transport.mode).

Procedure

1. Click Cloudera Manager Clusters HIVE_ON_TEZ Configuration
2. In Search, type transport.

3. In HiveServer2 Transport Mode, select http.

The screenshot shows the 'Hive on Tez' configuration interface. The 'Configuration' tab is active. A search bar contains 'transport'. The 'Filters' section shows 'SCOPE' with 'Hive on Tez (Service-Wide)' set to 1, 'Gateway' to 0, and 'HiveServer2' to 2. The 'HiveServer2 Transport Mode' section shows 'hive.server2.transport.mode' set to 'hive_server2_transport_mode'. The 'HiveServer2 Default Group' is set to 'binary'. The 'http' radio button is selected under the 'HiveServer2 Transport Mode' section. The 'Save Changes(CTRL+S)' button is visible at the bottom right.

4. Save and restart Hive on Tez.

Key syntax changes

You need to modify queries affected by changes to Hive syntax after upgrading to Cloudera. Hive has changed the syntax related to ``db.table`` references, such as `CREATE TABLE `mydb.mytable` ...`. Other syntax changes involve the `LOCATION` clause in `CREATE TABLE`. Hive in Cloudera supports the enhancement to `CREATE TABLE` that adds the `MANAGEDLOCATION` clause.

Handling table reference syntax

For ANSI SQL compliance, Hive 3.x rejects ``db.table`` in SQL queries as described by the Hive-16907 bug fix. A dot (.) is not allowed in table names. As a Data Engineer, you need to ensure that Hive tables do not contain these references before migrating the tables to Cloudera, that scripts are changed to comply with the SQL standard references, and that users are aware of the requirement.

About this task

To change queries that use such ``db.table`` references thereby preventing Hive from interpreting the entire `db.table` string incorrectly as the table name, you enclose the database name and the table name in backticks as follows:

A dot (.) is not allowed in table names.

Procedure

1. Find a table having the problematic table reference.
For example, `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));
```

LOCATION and MANAGEDLOCATION clauses

Before upgrading, your Hive version might have supported using the `LOCATION` clause in queries to create either managed or external tables or databases for managed and external tables. After upgrading, Hive stores managed and external tables in separate HDFS locations. `CREATE TABLE` limits the use of the `LOCATION` clause, and consequently requires a change to your queries. Hive in Cloudera also supports a new location-related clause.

External table limitation for creating table locations

Hive assigns a default location in the warehouse for external tables—/warehouse/tablespace/external/hive. In Cloudera, Hive does not allow the LOCATION clause in queries to create a managed table. Using this clause, you can specify a location only when creating external tables. For example:

```
CREATE EXTERNAL TABLE my_external_table (a string, b string)
ROW FORMAT SERDE 'com.mytables.MySerDe'
WITH SERDEPROPERTIES ( "input.regex" = "/*.csv" )
LOCATION '/warehouse/tablespace/external/hive/marketing' ;
```

Table MANAGEDLOCATION clause

In Cloudera, Hive has been enhanced to include a MANAGEDLOCATION clause to specify the location of managed tables as shown in the following syntax:

```
CREATE (DATABASE|SCHEMA) [IF NOT EXISTS] database_name
[COMMENT database_comment]
[LOCATION external_table_path]
[MANAGEDLOCATION managed_table_directory_path]
[WITH DBPROPERTIES (property_name=property_value, ...)] ;
```

Hive assigns a default location in the warehouse for managed tables—/warehouse/tablespace/managed/hive. In the MANAGEDLOCATION clause, you specify a top level directory for managed tables when creating a Hive database.



Important: Do not use the LOCATION clause to specify the location of managed tables. This clause is only used to specify the location of external tables. Use the MANAGEDLOCATION clause if you are creating managed tables. You must also ensure that you do not set LOCATION and MANAGEDLOCATION to the same HDFS path.

Use DESCRIBE DATABASE db_name; to view the root location of the database on the filesystem.

Related Information

[Create a default directory for managed tables](#)

Key semantic changes and workarounds

As SQL Developer, Analyst, or other Hive user, you need to know potential problems with queries due to semantic changes. Some of the operations that changed were not widely used, so you might not encounter any of the problems associated with the changes.

Over the years, Apache Hive committers enhanced versions of Hive supported in legacy releases of CDH and HDP, with users in mind. Changes were designed to maintain compatibility with Hive applications. Consequently, few syntax changes occurred over the years. A number of semantic changes, described in this section did occur, however. Workarounds are described for these semantic changes.

Changing incompatible column types

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

Before Upgrade to Cloudera

In HDP 2.x and CDH 5.x and CDH 6 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 to Cloudera

In Cloudera, 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. Check ALTER TABLE statements and change those that would fail due to incompatible column types.

Understanding CREATE TABLE behavior

Hive table creation has changed significantly since Hive 3 to improve useability and functionality. If you are upgrading from CDH or HDP, you must understand the changes affecting legacy table creation behavior.

Hive has changed table creation in the following ways:

- Creates ACID-compliant table, which is the default in Cloudera
- 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. Using ACID-compliant, transactional tables causes no performance or operational overload. The way you access managed Hive tables from Spark and other clients changes. In Cloudera, access to external tables requires you to set up security access permissions.

You must understand the behavior of the CREATE TABLE statement in legacy platforms like CDH or HDP and how the behavior changes after you upgrade to Cloudera.

Before upgrading to CDP Private Cloud Base

In CDH 5, CDH 6, and HDP 2, by default CREATE TABLE creates a non-ACID managed table in plain text format.

In HDP 3 and CDP 7.1.0 through 7.1.7.x, by default CREATE TABLE creates either a full ACID transactional table in ORC format or insert-only ACID transactional tables for all other table formats.

After upgrading to CDP Private Cloud Base

- If you are upgrading from HDP 2, CDH 5, or CDH 6 to CDP 7.1.0 through CDP 7.1.8, by default CREATE TABLE creates a full ACID transactional table in ORC format or insert-only ACID transactional tables for all other table formats.
- If you are upgrading from HDP 3 or CDP 7.1.0 through 7.1.7.x to CDP 7.1.8, the existing behavior persists and CREATE TABLE creates either a full ACID transactional table in ORC format or insert-only ACID transactional tables for all other table formats.

Now that you understand the behavior of the CREATE TABLE statement, you can choose to modify the default table behavior by configuring certain properties. The order of preference for configuration is as follows:

Modify the default CREATE TABLE behavior

Override default behavior when creating the table

Irrespective of the database, session, or site-level settings, you can override the default table behavior by using the MANAGED or EXTERNAL keyword in the CREATE TABLE statement.

```
CREATE [MANAGED][EXTERNAL] TABLE foo (id INT);
```

Set the default table type at a database level

You can use the database property, defaultTableType=EXTERNAL or ACID to specify the default table type to be created using the CREATE TABLE statement. You can specify this property when creating the database or at a later point using the ALTER DATABASE statement. For example:

```
CREATE DATABASE test_db WITH DBPROPERTIES ('defaultTableType'='EXTERNAL');
```

In this example, tables created under the test_db database using the CREATE TABLE statement creates external tables with the purge functionality enabled (external.table.purge = 'true').

You can also choose to configure a database to allow only external tables to be created and prevent creation of ACID tables. While creating a database, you can set the database property, EXTERNAL_TABLES_ONLY=true to ensure that only external tables are created in the database. For example:

```
CREATE DATABASE test_db WITH DBPROPERTIES ( 'EXTERNAL_TABLES_ONLY'='true' );
```

Set the default table type at a session level

You can configure the CREATE TABLE behavior within an existing beeline session by setting hive.create.as.external.legacy to true or false. Setting the value to true results in configuring the CREATE TABLE statement to create external tables by default.

When the session ends, the default CREATE TABLE behavior also ends.

Set the default table type at a site level

You can configure the CREATE TABLE behavior at the site level by configuring the hive.create.as.insert.only and hive.create.as.acid properties in Cloudera Manager under Hive configuration. When configured at the site level, the behavior persists from session to session. For more information, see [Configuring CREATE TABLE behavior](#).

If you are a Spark user, switching to legacy behavior is unnecessary. Calling 'create table' from SparkSQL, for example, creates an external table after upgrading to Cloudera as it did before the upgrade. You can connect to Hive using the Hive Warehouse Connector (HWC) to read Hive ACID tables from Spark. To write ACID tables to Hive from Spark, you use the HWC and HWC API. Spark creates an external table with the purge property when you do not use the HWC API. For more information, see [Hive Warehouse Connector for accessing Spark data](#).

Related Information

[HDFS ACLS](#)

[Hive Warehouse Connector for accessing Apache Spark data](#)

[Spark Direct Reader for accessing Spark data](#)

[Apache Hive 3 Key Features](#)

[Apache Hive 3 Tables](#)

Configuring legacy CREATE TABLE behavior

After you upgrade to CDP Private Cloud Base and migrate old tables, the legacy CREATE TABLE behavior of Hive is no longer available by default and you might want to switch to the legacy behavior. Legacy behavior might solve compatibility problems with your scripts during data migration, for example, when running ETL.

About this task

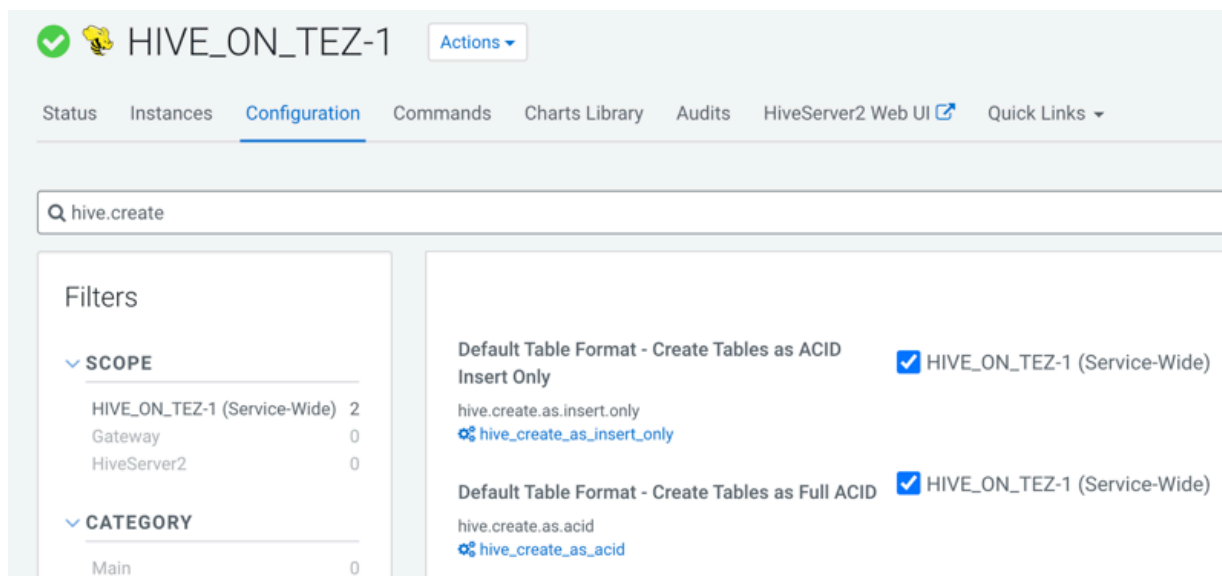
In Cloudera, running a CREATE TABLE statement by default creates a full ACID table for ORC file format and insert-only ACID table for other file formats. You can change the default behavior to use the legacy CREATE TABLE behavior. When you configure legacy behavior, CREATE TABLE creates external tables with the purge functionality enabled (external.table.purge = 'true'). Therefore, when the table is dropped, data is also deleted from the file system.

You can configure legacy CREATE TABLE behavior at the site level by configuring properties in Cloudera Manager. When configured at the site level, the behavior persists from session to session.

Procedure

1. In Cloudera Manager, click Clusters and select the Hive on Tez service.

- From the Hive on Tez service, go to the Configuration tab and search for hive.create.



- If the following properties are selected, clear the selection to enable legacy CREATE TABLE behavior.

- Default Table Format - Create Tables as ACID Insert Only (hive.create.as.insert.only)
- Default Table Format - Create Tables as Full ACID (hive.create.as.acid)

Results

Legacy behavior is enabled and the CREATE TABLE statement now creates external tables with the external.table.purge table property set to true.

Related Information

[Change DROP behavior](#)

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 to CDP Private Cloud Base

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

After Upgrade to CDP Private Cloud Base

OFFLINE and NO_DROP are not supported in the DROP CASCADE clause.

Action Required

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

Handling the Keyword APPLICATION

If you use the keyword APPLICATION in your queries, you might need to modify the queries to prevent failure.

To prevent a query that uses a keyword from failing, enclose the query in backticks.

Before Upgrade to CDP Private Cloud Base

In CDH releases, such as CDH 5.13, queries that use the word APPLICATION in queries execute successfully. For example, you could use this word as a table name.

```
> select f1, f2 from application
```

After Upgrade to Cloudera Base on premises

A query that uses the keyword APPLICATION fails.

Action Required

Change applications. Enclose queries in backticks. `SELECT field1, field2 FROM `application`;`

Handling output of greatest and least functions

To calculate the greatest (or least) value in a column, you need to work around a problem that occurs when the column has a NULL value.

Before Upgrade to Cloudera

The greatest function returned the highest value of the list of values. The least function returned the lowest value of the list of values.

After Upgrade to Cloudera

Returns NULL when one or more arguments are NULL.

Action Required

Use NULL filters or the `nvl` function on the columns you use as arguments to the greatest or least functions.

```
SELECT greatest(nvl(col1,default value incase of NULL),nvl(col2,default value incase of NULL));
```

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 to Cloudera

In CDH and HDP, renaming a managed table moves its HDFS location.

After Upgrade to Cloudera

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

TRUNCATE TABLE on an external table

Hive 3 does not support TRUNCATE TABLE on external tables. Truncating an external table results in an error. You can truncate an external table if you change your applications to set a table property to purge data.

Before Upgrade to Cloudera

Some legacy versions of Hive supported TRUNCATE TABLE on external tables.

After Upgrade to CDP Private Cloud Base

By default, TRUNCATE TABLE is supported only on managed tables. Attempting to truncate an external table results in the following error:

```
Error: org.apache.spark.sql.AnalysisException: Operation not allowed: TRUNCATE TABLE on external tables
```

Action Required

Change applications. Do not attempt to run TRUNCATE TABLE on an external table.

Alternatively, change applications to alter a table property to set `external.table.purge` to `true` to allow truncation of an external table:

```
ALTER TABLE mytable SET TBLPROPERTIES ('external.table.purge'='true');
```

Other syntax and semantic changes

In addition to the key Hive syntax and semantic changes, there are a number of other Hive changes in Cloudera you need to know about after migrating to Cloudera from CDH. The other changes are covered release by release.

Several of the other changes are the result of Cloudera being based on Apache Hive 3, which is closer to SQL standards than earlier versions of Hive, which CDH was based on.

The documentation includes a workaround if there is one.

Related Information

[Key syntax changes](#)

[Key semantic changes in Hive](#)

Syntax and semantic changes CDH 6.2.1 to CDP 7.0.3.2

Review the syntax and semantic changes in Hive after migrating to CDP 7.0.3.2 from CDH 6.2.1. A link to Apache Hive JIRAs, if there is one, provides more information about the semantic change.

Aliasing tables

Hive 3 in CDP follows the SQL 11 standard regarding a table identifier, which cannot begin with a numeral.

Before Upgrade to CDP

In CDH, using a table alias that begins with a numeral did not cause an error:

```
select * from db.t1 5d;
```

After Upgrade to CDP

In CDP, using a table alias that begins with a numeral causes an error:

```
select * from db.t1 5d;
```

Output looks something like this:

```
Error: Error while compiling statement: ...
```

Action Required

Perform either one of the following workarounds:

- Rename the table that begins with a numeral to conform to the SQL 11 standard.
- Enclose the table that begins with a numeral in backticks.

For more information, see [Handling table reference syntax](#).

ANALYZE TABLE ... COMPUTE STATISTICS PARTIALSCAN removed

The `analyze` command collects statistics on tables and partitions, but is slow when scanning many files. A command was added that operated on files in RCfile format to speed up statistics collection. The command was not widely used, not well documented, and eventually removed.

Before Upgrade to CDP

ANALYZE TABLE ... COMPUTE STATISTICS PARTIALSCAN was added to Hive to collect statistics, not by scanning all content of files, but by scanning only parts to get file metadata. Examples are shown in <https://cwiki.apache.org/confluence/display/Hive/RCFileCat>.

After Upgrade to CDP

[HIVE-17652](#) removes this syntax in release 3.0.0.

Action Required

In the unlikely event you used this command, remove it from your code.

Decimal to string change

Hive decimal to string conversion yields results having incorrect padding. LPAD on a decimal(7,1) value of 0 returns "0" (plus padding) but it should be "0.0" (plus padding).

Before Upgrade to CDP

Decimal to string conversion incorrectly formats the padding of the result.

```
select cast(cast('3404045.5044003' as decimal(18,9)) as string);
```

Output is: 3404045.5044003

After Upgrade to CDP

[HIVE-20082](#) corrected the problem. Output of the query above correctly pads the result to 9 digits after the decimal point: 3404045.504400300

Decimal literals

The default treatment of Decimal literals in Hive was changed from Double to Decimal as per the SQL standards.

Before upgrade to CDP

Decimal value is displayed as a rounded value when you run a SELECT statement where column = decimal value. For example,

```
create table test (dc decimal(38,18));
insert into table test values (4327269606205.029297);

select * from test;
Output:
+-----+-----+
|          test.dc          |
+-----+-----+
| 4327269606205.029297000000000000 |
+-----+-----+

select * from test where dc = 4327269606205.029297000000000000;
Output:
+-----+-----+
|          test.dc          |
+-----+-----+
| 4327269606205.029300000000000000 |
+-----+-----+
```

After upgrade to CDP

The fix changed the default treatment of Decimal literals in Hive and now prefers Decimal over Double.

This fix also influences other functions which accept such numeric values as parameters. For example:

```
select coalesce(2.0, 'EMPTY')
```


The coalesce functions' return type in this expression is string. In CDH, 2.0 was parsed as Java double and returned '2.0'. In CDP, it is parsed as HiveDecimal(1,0) and therefore returns '2'.

For more information, see [HIVE-13945](#).

hive.stats.collect.rawdatasize removal

In an early release of Hive, the hive.stats.collect.rawdatasize property collected statistics when set to true. As Hive matured, hive.stats.autogather was added to gather statistics when data is inserted into tables.

Before Upgrade to CDP

In Hive 0.8, [HIVE-2185](#) added the hive.stats.collect.rawdatasize to collect the raw data size of a table when analyzing the table.

After Upgrade to CDP

In Hive 2.1.0 the hive.stats.collect.rawdatasize property was removed ([HIVE-13564](#)).

HIVE_SUPPORT_SQL11_RESERVED_KEYWORDS

To simplify the parser logic and largely reduce the size of generated parser code, the HIVE_SUPPORT_SQL11_RESERVED_KEYWORDS configuration has been removed.

Before Upgrade to CDP

[HIVE-14872](#) removed support for this configuration in Hive 2.3.

HIVE_SUPPORT_SQL11_RESERVED_KEYWORDS configuration offered backward compatibility. Reserved keywords were used as identifiers in the previous releases.

After Upgrade to CDP

Cloudera releases support later versions of Hive that do not support the configuration.

Limit scanned partitions

Before Hive 1.0 a configuration property was added for limited scanned partitions that was replaced by another property.

Before Upgrade to CDP

To protect the cluster, the hive.limit.query.max.table.partition configuration property was added to Hive. The property limits the table partitions involved in a table scan. In Hive 2.2.0, the hive.metastore.limit.partition.request was added to replace the hive.limit.query.max.table.partition. The hive.limit.query.max.table.partition support was deprecated.

After Upgrade to CDP

In Hive 3, support for hive.limit.query.max.table.partition was removed ([HIVE-17965](#)). Only replacement [hive.metastore.limit.partition.request](#) is supported to limit the number of partitions that can be requested from the Metastore for a table. A query is not executed if it fetches more partitions per table than the limit configured. A value of -1 means unlimited.

Action Required

Find any settings of hive.limit.query.max.table.partition and change your code to use hive.metastore.limit.partition.request.

Overflow handling of decimals

An ArrayIndexOutOfBoundsException is returned when a sum operation on a decimal column overflows beyond the maximum precision.

Before upgrade to CDP

When a column is defined as decimal and a sum operation on that column overflows beyond the maximum decimal precision (38), the an exception occurs:

```
create table decimal_precision(dec decimal(38,18));
insert into decimal_precision values(98765432109876543210.12345), (98765432109876543210.12345);
```

```
select sum(dec) from decimal_precision;
```

This query fails with an `ArrayIndexOutOfBoundsException`.

After upgrade to CDP

[HIVE-13423](#) handles this issue by displaying a warning message and returning NULL when the result of a SUM operation on a decimal column goes beyond the decimal precision range.

Functions that changed

A number of changes to Hive functions occurred in CDP.

When creating a UDF, you must extend the `GenericUDF` class. For more information, see [Creating the UDF](#) class. A number of changes to built-in functions described in this documentation also occurred.

ACOS(2) and ASIN(2) return NULL

A change in the result of `ACOS(2)` and `ASIN(2)` occurred between CDH and CDP.

Before Upgrade to CDP

The `ACOS(2)` and `ASIN(2)` functions returned NAN as shown in the following example:

```
select acos(2);
NaN
```

After Upgrade to CDP

[HIVE-17240](#) makes the `ACOS(2)` and `ASIN(2)` functions return NULL, consistent with SQL standards.

```
select acos(2);
NULL
```

CAST function results

When casting a decimal value which has only zeros in the fractional digits, the `CAST` function in CDP differs from CDH.

Before Upgrade to CDP

In CDH, fractional digits appear in the output of a cast of a decimal value that has only zeros in the fractional part.

For example, run this query that implicitly casts a number having insignificant decimal thousandths:

```
SELECT 123.000
```

The output is a 0 tenths decimal: 123.0.

Explicit casts produce similar incorrect results.

```
select cast(123.000 as varchar(10)) ;
```

The output is a 0 tenths decimal: 123.0.

Hive behavior is the same when selecting data from a table.

After Upgrade to CDP

In CDP, fractional decimal digits are dropped in the output of a cast of a decimal value that has only zeros in the fractional part ([HIVE-15335](#)).

```
SELECT 123.000;
```

The output is the whole number: 123.

Hive behavior is the same when you explicitly cast columns and select data from a table.

Action Required

To get the legacy CDH behavior in CDP, cast as follows:

```
select cast('123.000' as string) ;
```

Casting types with leading or trailing spaces

Learn about the fix to resolve inconsistent CAST behavior while casting string to numeric data types that have leading or trailing whitespace characters.

Before upgrade to CDP

The CAST function resulted in incorrect results while casting string values with leading or trailing spaces to numeric data types. For example,

```
select cast(' 1 ' as tinyint), cast(' 1 ' as smallint), cast(' 1 ' as int),
cast(' 1 ' as bigint), cast(' 1 ' as float), cast(' 1 ' as double), cast(' 1
' as decimal(10,2))
```

Output:

NULL	NULL	NULL	NULL	1.0	1.0	1
------	------	------	------	-----	-----	---

The numeric data types, such as tinyint, smallint, int, and bigint were not converted correctly because of leading and trailing spaces and returned NULL. However, float, double, and decimal returned the correct output.

After upgrade to CDP

[HIVE-17782](#) provides the fix to ensure consistent cast behavior across data types. The fix trims leading or trailing spaces in the string value before passing the value to the number formatter.

CORR and COVAR_SAMP compliant with SQL:2011

Learn about the changes introduced to ensure that the CORR and COVAR_SAMP built-in aggregate functions (UDAFs) are in compliance with the SQL:2011 standards.

Before upgrade to CDP

The CORR and COVAR_SAMP aggregate functions were not in compliance with SQL:2011 standards. CORR returned 'NaN' for $N * \text{SUM}(x * x) = \text{SUM}(x) * \text{SUM}(x)$ and $N * \text{SUM}(y * y) = \text{SUM}(y) * \text{SUM}(y)$. However, the function is expected to return NULL per the standards.

Similarly, COVAR_SAMP returns '0' when the function is applied to a set with a single element. The function is expected to return NULL.

After upgrade to CDP

[HIVE-16178](#) fixes the compliance issues and ensures that these functions comply with SQL:2011 standards.

LENGTH function supported data types

The data types supported by the LENGTH function in CDP differ from CDH. The behavioral difference presents a problem when using the absolute value function ABS.

Before Upgrade to CDP

In CDH, the LENGTH function supports double in addition to string, char, varchar or binary. For example, the following query is valid:

```
select length(abs('123.000'));
```

After Upgrade to CDP

In CDP, [HIVE-15979](#) is implemented to follow the SQL standard. LENGTH supports columns of data type string, char, varchar or binary. Double is not supported by length() in CDP. For example, an error occurs when passing a double to the abs function.

```
select length(abs('123.000'));
```

```
Error: Error while compiling statement: FAILED: SemanticException [Error 10014]: Line 1:7 Wrong arguments '123.000': LENGTH() only takes STRING/CHAR/VARCHAR/BINARY types as first argument ...42000
```

For more information, see the [Generic UDF Length definition](#).

Action Required

In CDP, use the workaround shown in the following example:

```
select length(cast(abs('123.000') as char(10)));
```

STDDEV_SAMP and VAR_SAMP

Learn about the changes introduced to ensure that the STDDEV_SAMP and VAR_SAMP built-in aggregate functions (UDAFs) are in compliance with the SQL:2011 standards.

Before upgrade to CDP

The STDDEV_SAMP and VAR_SAMP aggregate functions were not in compliance with SQL:2011 standards and returns '0' when the function is applied to a set with a single element. The functions are expected to return NULL.

After upgrade to CDP

[HIVE-17375](#) fixes the issue and ensures that these functions comply with SQL:2011 standards. The functions now return NULL when they are applied to either an empty set or a set with a single element.

NULL related behaviors

Several behavior changes are due to the handling of NULLs.

ORDER BY clause treatment of NULLs

The default value of hive.default.nulls.last in CDP differs from the default value in CDH, which causes a behavioral difference in the ORDER BY clause, including ORDER BY within a RANK function.

Before Upgrade to CDP

In CDH, hive.default.nulls.last is false. In ascending (ASC) order, NULL appears first. In descending (DESC) order, NULL appears last.

After Upgrade to CDP

In CDP, hive.default.nulls.last is true. In ascending (ASC) order, NULL appears last. In descending (DESC) order, NULL appears first ([HIVE-23706](#)),

Action Required

In CDP, set hive.default.nulls.last to false for legacy behavior. For more information, see [Hive default null sorting order for DESC is NULL FIRST](#).

Disallow enabling/enforcing NOT NULL

Hive does not manage the data for external tables and cannot enforce constraints on these tables.

Before Upgrade to CDP

You could enable/enforce a NOT NULL constraint on an external table. For example:

```
CREATE EXTERNAL TABLE t(a TINYINT, b SMALLINT NOT NULL ENABLE, c INT);
```

After Upgrade to CDP

Enabling or enforcing a NOT NULL constraint on an external table causes an error ([HIVE-18598](#)).

Action Required

Prevent errors by removing the NOT NULL constraint in CREATE TABLE or ALTER TABLE statements.

Default NULL ordering change

The default NULL ordering changed from CDH to Cloudera. A workaround can prevent a possible performance hit.

Before Upgrade to CDP

In CDH, [HIVE-20423](#) set NULLS LAST to be default ordering.

After Upgrade to CDP

In CDP, NULLS FIRST is the default ordering.

Action Required

To prevent the change from affecting your workload performance, such as TopNKey operations, follow instructions in the [Knowledge Base](#) to set NULLS LAST as the default.

Enforcement of NOT NULL constraint

Before Upgrade to CDP

Hive did not support the enforcement of NOT NULL.

After Upgrade to CDP

[HIVE-16605](#) supports NOT NULL constraint enforcement on INSERT, MERGE, and UPDATE statements. This capability is configurable by setting the hive.constraint.notnull.enforce property. For more information, see [Using constraints](#).

Timestamp or date related behaviors

A few behavior changes are related to timestamp and date formats, functions, and data types.

ADD_MONTHS function fix

The output of the ADD_MONTHS function on a timestamp column was corrected to include the time.

Before Upgrade to CDP

The output of the following query incorrectly omitted the time:

```
select CUSTOMER_ID,EMAIL_FAILURE_DTMZ,ADD_MONTHS (EMAIL_FAILURE_DTMZ , 1) fr
om TABLE1 where CUSTOMER_ID=125674937;
```

Example output:

```
1125674937    2015-12-09 12:25:53    2016-01-09
```

After Upgrade to CDP

[Hive-19370](#) fixed the omission, so the output of the query above includes the time.

Example output:

```
125674937    2023-12-09 12:25:53    2024-01-09 12:25:53
```

ADD_MONTHS date validation

A change, which is not likely to cause any migration problems, was made to the ADD_MONTHS function.

Before Upgrade to CDP

The ADD_MONTHS function would execute on invalid dates, such as 2017-02-29.

After Upgrade to CDP

[Hive-18746](#) performs validation of the date before executing the ADD_MONTHS function. The output indicates an invalid date as shown below:

```
select add_months('2017-02-29', 1);
'2017-02-29' is an invalid date.
```

Casting invalid dates

Casting of an invalid date differs from Hive 1 in CDH 5 to Hive 3 in CDP. Hive 3 uses a different parser formatter from the one used in Hive 1, which affects semantics. Hive 1 considers 00 invalid for date fields. Hive 3 considers 00 valid for date fields. Neither Hive 1 nor Hive 3 correctly handles invalid dates, and Hive-25056 addresses this issue.

Before Upgrade to CDP

Casting of invalid date (zero value in one or more of the 3 fields of date, month, year) returns a NULL value:

```
SELECT CAST ('0000-00-00' as date) , CAST ('000-00-00 00:00:00' AS TIMESTAMP) ;
```

After Upgrade to CDP

Casting of an invalid date returns a result.

```
> SELECT CAST ('0000-00-00' as date) , CAST ('000-00-00 00:00:00' AS TIMESTAM
MP) ;
...
00002-11-30 00:00:00.0
```

Action Required

Do not cast invalid dates in Hive 3.

FROM_UNIXTIME and UNIX_TIMESTAMP time zone

The FROM_UNIXTIME and UNIX_TIMESTAMP functions have undergone more than one time zone change.

Before Upgrade to CDP

The functions supported the system time zone.

After Upgrade to CDP

The functions now support the user session time zone ([HIVE-22170](#)).

Handling of CURRENT_TIMESTAMP output format

The output format of the CURRENT_TIMESTAMP user defined function (UDF) is inconsistent in certain scenarios.

Before upgrade to CDP

CURRENT_TIMESTAMP returns the result in different output formats. For example,

```
select current_timestamp();
2016-04-14 18:26:58.875
select current_timestamp() from all100k union select current_timestamp() fr
om over100k limit 5;
2016-04-14 18:29:56
```

In the second query, the fractional seconds precision is removed from the output.

After upgrade to CDP

[HIVE-13837](#) fixes this issue and the output format for CURRENT_TIMESTAMP YYYY-MM-DD HH:MM:SS.fff.

Handling of Julian dates in UDFs

The way Julian dates are handled in CDP is improved over the way CDH handled these dates.

Before Upgrade to CDP

Julian calendar (before Oct 15, 1582) dates are handled improperly by date/timestamp UDFs.

For example, the DateFormat UDF behavior is as follows:

Dates in the Julian calendar in your input are misinterpreted as Gregorian calendar dates. A multiple-day error can occur. For example:

```
beeline> select date_format('1001-01-05','dd---MM--yyyy');
+-----+
| _c0 |
+-----+
| 30---12--1000 |
+-----+
```

Problems have been reported in the following UDFs:

- add_months
- date_format
- day
- month
- months_between
- weekofyear
- year

After Upgrade to CDP

In CDP, Hive uses a proleptic Gregorian calendar used in SQL standard to handle Julian dates ([Hive-22099](#)).

Handling return type for old date functions

Learn about the changes related to the return type for some of the originally introduced date functions. The return type is changed from string to date.

Before upgrade to CDP

Some of the original date functions, such as TO_DATE, DATE_ADD, and DATE_SUB returned string values instead of date values. This was because the date return type was not available in Hive when these functions were introduced.

After upgrade to CDP

[HIVE-13248](#) changes the return type of to_date, date_add, and date_sub functions from string to date.

Support for SQL:2016 datetime formats (limited formats)

This change introduces support for the SQL:2016 FORMAT clause for CAST which is the most widely used method to perform string to datetime conversions. The change also includes support for a limited list of SQL:2016 datetime formats.

Before upgrade to CDP

Timestamp/date handling and formatting is currently implemented in Hive using the [Java SimpleDateFormat patterns](#), however, this is not what most standard SQL systems use. For example see [Oracle](#) and [PostgreSQL](#).

After upgrade to CDP

CDP includes support for datetime format patterns as recommended by the ISO and ANSI SQL:2016 standard for SQL database query language. SQL:2016 also includes the FORMAT clause for CAST function, which is the preferred way to perform string to datetime conversions.

Usage

```
CAST(<datetime> AS <char string type> [FORMAT <template>])
CAST(<char string> AS <datetime type> [FORMAT <template>])
```

Examples

```
cast(dt as string format 'DD-MM-YYYY')
cast('01-05-2017' as date format 'DD-MM-YYYY')
```

For more information about these patterns, see [CAST...FORMAT with SQL:2016 datetime formats](#).

For more information about the change, see [HIVE-21576](#).

Action required

None. There is no configuration or feature flag introduced by this change to start using the new SQL standard formats. Legacy functions, such as TO_TIMESTAMP and FROM_TIMESTAMP will continue to follow the SimpleDateFormat patterns and CAST (...FORMAT...) will use the SQL:2016 patterns.

UNIX_TIMESTAMP behavior

The behavior of the UNIX_TIMESTAMP function in CDP differ in several ways from CDH.

Before Upgrade to CDP

- In CDH, you can use lowercase hh to represent hours of a timestamp in the format specification.
- You do not have to include microseconds in the format specification. The format MM-dd-yyyy HH:mm:ss is sufficient.
- Casting a unix timestamp as a string works.

Example of using hh, which works:

```
select unix_timestamp('2024-12-30 59:10:20', "yyyy-MM-dd hh:mm:ss"), from_utc_timestamp(from_unixtime(unix_timestamp('2024-12-30 59:10:20', "yyyy-MM-dd hh:mm:ss")), 'yyyy-MM-dd HH:mm:ss'), 'CST'), from_utc_timestamp(from_unixtime(unix_timestamp('2024-12-30 T59:10:20.192+0000', "yyyy-MM-dd 'T' hh:mm:ss.SSS' + 0000"), 'yyyy-MM-dd HH:mm:ss'), 'CST') ;
```

Example of missing microseconds, which works.

```
select unix_timestamp('12-12-2023 15:30:12.075', 'MM-dd-yyyy HH:mm:ss');
```

Example of casting timestamp as a string, which works:

```
select cast(unix_timestamp('2023-04-03:10:10:00', 'yyyyMM') as string);
```

After Upgrade to CDP

- Hive in CDP requires uppercase HH and microseconds SSS are required in the format specification. For example, 'MM-dd-yyyy HH:mm:ss.SSS'; otherwise, output is NULL.
- Casting a timestamp yields NULL.

Example of using hh

Action Required

Change code from hh to HH and define microseconds.

```
select unix_timestamp('11-11-2020 15:30:12.084', 'MM-dd-yyyy HH:mm:ss.SSS');
```

For more information about timestamp issues, see [HIVE-25458](#).

TIMESTAMP based on UTC

The Hive TIMESTAMP in CDH was not UTC-based. CDP supports UTC-based, SQL TIMESTAMP.

Greenwich Mean Time (GMT), also known as Coordinated Universal Time, Universal Time Coordinated (UTC), Z time, or Zulu time is a coordinated time scale managed by the Bureau International des Poids et Mesures (BIPM).

Before Upgrade to CDP

Hive used the local time (java.sql.Timestamp) to represent the `TIMESTAMP` data type without a time zone. This behavior misrepresented the time in some zones.

After Upgrade to CDP

UTC is used instead of the local time to represent the `TIMESTAMP` data type without a time zone ([HIVE-12192](#)).

UNIX_TIMESTAMP conversion of TIMESTAMPTOLOCALTZ

CDP corrects an issue in CDH related to the `UNIX_TIMESTAMP` conversion of a local time zone.

Before Upgrade to CDP

Attempting to convert a timestamp of data type `TIMESTAMPTOLOCALTZ` using `UNIX_TIMESTAMP`, as shown in the following example causes an exception.

```
set hive.local.time.zone=Asia/Bangkok;

SELECT FROM_UNIXTIME(UNIX_TIMESTAMP('2000-01-07 00:00:00 GMT','yyyy-MM-dd HH:mm:ss z'));
```

The error looks something like this:

```
org.apache.hadoop.hive.ql.parse.SemanticException: Line 3:456 Wrong argument
s 'yyyy-MM-dd HH:mm:ss': The function UNIX_TIMESTAMP takes only string/dat
e/timestamp types
```

After Upgrade to CDP

[HIVE-18595](#) fixed the issue described above.

Action Required

Before upgrading from CDH to CDP, remove any conversions of the local time zone timestamp using `UNIX_TIMESTAMP`.

Semantic changes and workarounds CDP 7.1.1

Review the semantic changes in Hive after migrating to CDP 7.1.1 from CDP 7.0.3.2. A link to Apache Hive JIRAs, if there is one, provides more information about the semantic change. If you are migrating to CDP 7.1.1 from CDH 6.2.1, review the list of Hive changes in each CDP release prior to CDP 7.1.1.

NVL UDF implementation changes

The `NVL` user defined function (UDF) is used to replace a `NULL` value with any value other than `NULL`. The implementation of the `NVL` function is the same as the implementation of the `COALESCE` function and therefore is retired.

Before upgrade to 7.1.1

Prior to this change, the `NVL` and `COALESCE` function were both used for the same purpose - to replace a `NULL` value with a non-`NULL` value. Both the functions were compiled separately.

After upgrade to 7.1.1

With this change, the `NVL` function is compiled to `COALESCE`, which is reflected in the execution plan generated using the `EXPLAIN` command. The optimizations that are performed on the `COALESCE` function are also performed on the `NVL` calls.

For more information, see [HIVE-20961](#).

Improved Handling of External Table Inserts in HDFS

This change fixes an issue where empty file gets created when `INSERT OVERWRITE` is executed.

Before upgrade to CDP 7.1.1

When creating an EXTERNAL table and executing `INSERT OVERWRITE`, an empty file gets created in the HDFS storage location.

After upgrade to CDP 7.1.1

Starting with CDP Private Cloud Base 7.1.1, no empty files are created in the HDFS location when executing `INSERT OVERWRITE` on an external table because of a fix included in [HIVE-22941](#).

This improvement, enhances Hive's performance, benefits other systems, and avoids the generation of small files in HDFS.

Semantic changes and workarounds CDP 7.1.4

Review the semantic changes in Hive after migrating to CDP 7.1.4 from CDP 7.1.3. A link to Apache Hive JIRAs, if there is one, provides more information about the semantic change. If you are migrating to CDP 7.1.4 from CDH 6.2.1, review the list of Hive changes in each CDP release prior to CDP 7.1.4.

Exclusive write lock for MERGE INSERT

Learn how you can enable EXCLUSIVE (X) lock for the MERGE INSERT operations, which helps in preventing duplicates during concurrent MERGE INSERT operations.

Before upgrade to CDP 7.1.4

MERGE INSERT operation is treated as regular INSERT and acquires a SHARED_READ lock, which does not prevent other INSERT operations. Two concurrent MERGE INSERT operations generate duplicates due to the lack of locking.

After upgrade to CDP 7.1.4

A new configuration, `hive.txn.xlock.mergeinsert` is introduced to ensure that MERGE INSERT operations acquire EXCLUSIVE or EXCL_WRITE lock for transactional tables.

When this property is set to 'true', it prevents duplicates when MERGE INSERT statements are running in parallel transactions. By default, the property is set to 'false'.

If `hive.txn.xlock.write=false` (optimistic concurrency control), EXCL_WRITE is acquired by MERGE INSERT and SHARED_WRITE is acquired by INSERT, else the operations acquire EXCLUSIVE and SHARED_READ respectively. Both of these combinations prevent concurrent execution.

Action required

Ensure that you enable `hive.txn.xlock.mergeinsert` if you want to provide EXCLUSIVE lock for the MERGE INSERT operation.

For more information, see [HIVE-24000](#).

Lock implementations to allow zero-wait readers

Learn about the implementation changes introduced in locks to allow zero-wait or no-wait readers. You can enable certain properties to ensure that SHARED_READ does not have to wait for any lock and can fail immediately for a pending EXCLUSIVE DDL operation.

Before upgrade to CDP 7.1.4

By default, the implementation changes as part of this fix are disabled and read operations have to wait to acquire a lock.

After upgrade to CDP 7.1.4

This change introduces a new EXCL_WRITE lock type that improves concurrency of transactions by moving some of the operations to another operation like INSERT OVERWRITE that has a less restrictive lock.

This ensures that regular INSERT INTO operations (with a SHARED_WRITE locktype) that are running concurrently are not hidden by the INSERT OVERWRITE operation (with a EXCL_WRITE locktype).

The new implementation is auto-enabled when either the `hive.txn.xlock.write` or `hive.txn.xlock.iow` properties are set to 'false'. This ensures that read operations fail immediately if there is a DDL operation like DROP with EXCLUSIVE lock type.

Action required

Enable the new implementation by setting either `hive.txn.xlock.write` or `hive.txn.xlock.iow` to 'false'.

For more information, see [HIVE-19369](#).

UNBOUNDED representation in Window functions

Learn about the implementation change for the ROWS clause of the Window function. The boundary specification has now been updated to consider the UNBOUNDED representation.

Before upgrade to CDP 7.1.4

The implementation of the ROWS clause in Window functions previously considered a bounded representation with the limits set to `Integer.MAX_VALUE` (2147483647).

After upgrade to CDP 7.1.4

The bounded representation was changed to use UNBOUNDED to align with the `RexWindowBound` boundary class of the Apache Calcite SQL parser.

For more information, see [HIVE-23275](#).

Support for 0 ROWS PRECEDING or FOLLOWING

Learn about the change that provides support for 0 ROWS PRECEDING or ROWS FOLLOWING in Window function specifications.

Before upgrade to CDP 7.1.4

The support for '0' as a Window frame boundary in ROWS PRECEDING or ROWS FOLLOWING was removed as part [HIVE-12574](#).

After upgrade to CDP 7.1.4

The Window function specification is enhanced to support a Window frame boundary value of '0' for ROWS PRECEDING and ROWS FOLLOWING, and replaces it with CURRENT ROW in the query plan. The behavior is the same as using 0 ROWS PRECEDING/FOLLOWING.

For more information, see [HIVE-23868](#).

Semantic changes and workarounds CDP 7.1.5

Review the semantic changes in Hive after migrating to CDP 7.1.5 from CDP 7.1.4. A link to Apache Hive JIRAs, if there is one, provides more information about the semantic change. If you are migrating to CDP 7.1.5 from CDH 6.2.1, review the list of Hive changes in each CDP release prior to CDP 7.1.5.

Sort behavior in SHOW COLUMNS

Learn about the change in sorting behavior when you run the SHOW COLUMNS statement. The resulting output is not sorted unless explicitly specified.

Before upgrade to CDP 7.1.5

Previously, the default behavior of the SHOW COLUMNS statement displayed the output in a sorted manner.

Example:

```
CREATE TABLE foo (c INT, a INT, b INT);
SHOW COLUMNS in foo;
```

Output:

```
a
b
c
```

Expected output:

```
c
a
b
```

After upgrade to CDP 7.1.5

The default behavior of the SHOW COLUMNS statement was changed to display the columns as is without sorting. If you want the output to be sorted, you must provide the optional keyword 'SORTED'.

Example: For the table created in the above example, here is the modified SHOW COLUMNS behavior:

```
SHOW COLUMNS in foo;
```

Output:

```
c
a
b
```

```
SHOW SORTED COLUMNS in foo;
```

Output:

```
a
b
c
```

For more information, see [HIVE-24282](#).

Event notification cleanup interval

A new configuration is introduced to specify the Time-to-live (TTL) for event notifications based on whether replication is enabled. The TTL determines how long the events can remain in the Hive Metastore (HMS).

Before upgrade to CDP 7.1.5

The `hive.metastore.event.db.listener.timetolive` property was used to determine how long events are stored in the HMS after which the events are removed from the database listener queue. The default value for this property is set to 1 day. However, this does not take replication into context in order to have a longer duration.

After upgrade to CDP 7.1.5

A new property, `hive.repl.event.db.listener.timetolive` is introduced to determine how long events are stored in the HMS based on whether replication (`hive.repl.cm.enabled`) is enabled or not. The default value of this property is set to 10 days.

When `hive.repl.cm.enabled` is set to true, the TTL is determined by the value specified in `hive.repl.event.db.listener.timetolive`. Else, if the value is set to false, the TTL reverts to the earlier behavior and is determined by the value specified in `hive.metastore.event.db.listener.timetolive`.

For more information, see [HIVE-24173](#).

Action required

If replication is enabled (`hive.repl.cm.enabled=true`), then ensure that the `hive.repl.event.db.listener.timetolive` property is set to the required time for which you want event notifications to be stored in the HMS.

Semantic changes and workarounds CDP 7.1.6

Review the semantic changes in Hive after migrating to CDP 7.1.6 from CDP 7.1.5. A link to Apache Hive JIRAs, if there is one, provides more information about the semantic change. If you are migrating to CDP 7.1.6 from CDH 6.2.1, review the list of Hive changes in each CDP release prior to CDP 7.1.6.

Support for SQL:2016 datetime formats (text, FM, FX)

This change enables Hive to parse the SQL:2016 datetime formats (text, FM, and FX) when a combination or subset of these formats or any of the previously implemented formats are provided in a string.

Before upgrade to CDP

Support was available for a limited set of SQL:2016 datetime formats introduced as part of [HIVE-21576](#).

After upgrade to CDP

CDP 7.1.6 includes support for additional datetime format patterns as recommended by the ISO and ANSI SQL:2016 standard for SQL database query language. The list of datetime formats added as part of this change are as follows:

- Nested strings (Text)
- Fill mode modifier (FM)
- Format exact modifier (FX)

For more information about these patterns, see [CAST...FORMAT with SQL:2016 datetime formats](#).

For more information about the change, see [HIVE-21578](#).

Action required

None. There is no configuration or feature flag introduced by this change to start using the new SQL standard formats. Legacy functions, such as `to_timestamp()` and `from_timestamp()` will continue to follow the SimpleDateFormat patterns and `CAST (...FORMAT...)` will use the SQL:2016 patterns.

Casting Timestamp to numeric and vice-versa

Casting a Timestamp to a numeric, or a numeric to a Timestamp, is not allowed by default and not supported by the SQL Standard.

Before Upgrade to CDP 7.1.6

In CDP 7.1.5 and earlier, casting to/from Timestamp/numeric was allowed.

After Upgrade to CDP 7.1.6

In CDP 7.1.6, casting to/from Timestamp/numeric is not allowed by default. A new `hive.strict.timestamp.conversion` property was introduced and set to true by default.

```
select cast(123 as timestamp);
```

Output is:

```
FAILED: SemanticException Line 0:-1 Wrong arguments '123': Casting NUMERIC types to TIMESTAMP is prohibited (hive.strict.timestamp.conversion)
```

For more information, see [HIVE-24157](#).

Action Required

To use the legacy behavior that allows casting a Timestamp to numeric and vice-versa, set `hive.strict.timestamp.conversion` to false.

Handling trailing zeros of decimal constants

Learn about the change that ensures that decimal constants are padded with trailing zeros according to the specified scale.

Before upgrade to CDP 7.1.6

Hive removes trailing zeros of decimal constants in some cases. Padding decimal values with trailing zeros is not consistent. For example,

```
select cast(cast(1.1 as decimal(22, 2)) as string), cast(cast(sum(1.1) as decimal(22, 2)) as string)
```

```
Output:
1.1      1.10
```

After upgrade to CDP 7.1.6

[HIVE-24389](#) provides the fix that pads constant decimal values with trailing zeros up to the specified scale.

Semantic changes and workarounds CDP 7.1.7

Review the semantic changes in Hive after migrating to CDP 7.1.7 from CDP 7.1.6. A link to Apache Hive JIRAs, if there is one, provides more information about the semantic change. If you are migrating to CDP 7.1.7 from CDH 6.2.1, review the list of Hive changes in each CDP release prior to CDP 7.1.7.

Precision and scale changes

Learn about the change in behavior of precision and scale in arithmetic operations.

Before Upgrade to CDP 7.1.6

In CDP Private Cloud Base 7.1.5 and lower versions, the default maximum precision and scale for arithmetic operations was calculated differently. For example, the following expression returns a result of 1523746134454743 7170.339101086652 with decimal(34, 12):

```
select cast( '12346789101112123.12345678' as decimal(25,8)) * cast( '1234.1234' as decimal(25,8))
```

After Upgrade to CDP 7.1.6

In CDP Private Cloud Base 7.1.6 and higher versions, the default maximum precision and scale for arithmetic operations follow the specifications defined in [Microsoft SQL Server documentation](#).

As a result, the above expression returns a result of 15237461344547437170.339101 with decimal(38, 6).

For more information, see [HIVE-24389](#), [HIVE-25263](#), and [HIVE-15331](#).

Semantic changes and workarounds CDP 7.1.7 SP1

Review the semantic changes in Hive after migrating to CDP 7.1.7 SP1 from CDP 7.1.7. A link to Apache Hive JIRAs, if there is one, provides more information about the semantic change. If you are migrating to CDP 7.1.7 SP1 from CDH 6.2.1, review the list of Hive changes in each CDP release prior to CDP 7.1.7 SP1.

Date and timestamp parser changes from LENIENT to STRICT

Learn about the changes introduced in DateTimeFormatter class, which is used to parse string, int, and char into Date or Timestamp.

Before Upgrade to CDP 7.1.7 SP1

Before the upgrade, the DateTimeFormatter class was set to ResolverStyle.LENIENT, which results in converting an incorrect date or timestamp instead of NULL. For example, "1992-13-12" was converted to "2000-01-12".

After Upgrade to CDP 7.1.7 SP2

The DateTimeFormatter class was enhanced to use the ResolverStyle.STRICT, which returns NULL when an incorrect date or timestamp is casted.

For more information, see [HIVE-25306](#).

Date strings are parsed using local timezone

Certain UDFs, such as datefiff() that use the VectorUDFDateDiffColScalar class, return an incorrect result when parsing scalar dates. Learn about the changes introduced to fix this issue.

Before Upgrade to CDP 7.1.7 SP1

The VectorUDFDateDiffColScalar class uses java.text.SimpleDateFormat to parse the date strings and interpret the scalar date to be in the local timezone.

Example:

```
create external table test_dt(id string, dt date);
```

```
insert into test_dt values('11', '2021-07-06'), ('22', '2021-07-07');

select datediff(dt1.dt, '2021-07-01') from test_dt dt1 left join test_dt dt
on dt1.id = dt.id;
```

Output:

```
+-----+
| _c0    |
+-----+
| 6      |
| 7      |
+-----+
```

Expected output:

```
+-----+
| _c0    |
+-----+
| 5      |
| 6      |
+-----+
```

After Upgrade to CDP 7.1.7 SP1

The parsing mechanism of the VectorUDFDateDiffColScalar class is updated to interpret date strings in Coordinated Universal Time (UTC) and the UDFs now return correct values.

For more information, see [HIVE-25449](#).

Semantic changes and workarounds CDP 7.1.7 SP2

Review the semantic changes in Hive after migrating to CDP 7.1.7 SP2 from CDP 7.1.7 SP1. A link to Apache Hive JIRAs, if there is one, provides more information about the semantic change. If you are migrating to CDP 7.1.7 SP2 from CDH 6.2.1, review the list of Hive changes in each CDP release prior to CDP 7.1.7 SP2.

Date and timestamp format changes

Learn about the change in the way date and timestamp values are parsed.

Before Upgrade to CDP 7.1.7 SP2

Some of the Hive date and timestamp functions used the SimpleDateFormat class for formatting and parsing date and timestamp. For more information, refer to the [SimpleDateFormat class Javadocs](#).

After Upgrade to CDP 7.1.7 SP2

The following Hive date and timestamp functions are now enhanced to use the DateTimeFormatter class for printing and parsing date and timestamp objects. For more information, refer to the [DateTimeFormatter class Javadocs](#).

- `unix_timestamp()`: This function is enhanced to use the DateTimeFormatter class for String format dates instead of the SimpleDateFormat class. For details, see [HIVE-25458](#).
- `from_unixtime()`: This function is now enhanced to consider leap seconds. For details, see [HIVE-25403](#).
- `date_format()`: This function that previously returned the output in UTC time zone is enhanced to display the default user session time zone. For details, see [HIVE-25093](#).
- `cast()`: This function is enhanced to display NULL when an incorrect date or timestamp is casted. Prior to this enhancement, when an incorrect date was casted, the function returned a converted value. For example, cast ('2020-20-20' as date) resulted in '2021-08-20' instead of NULL.

This is because the DateTimeFormatter class that is used to parse string into date or timestamp was set to ResolverStyle.LENIENT. This is now updated to use ResolverStyle.STRICT and returns NULL when an invalid date or timestamp is casted. For details, see [HIVE-25306](#).

**Important:**

Two identical patterns can be interpreted differently by the `SimpleDateFormat` and `DateTimeFormatter` class leading to different results. Even if the same letters appear in the javadoc of both the classes, their semantics might be different. Therefore, it is important that you read the javadoc carefully to understand the behavior.

For example, consider the pattern "DD". In both `SimpleDateFormat` and `DateTimeFormatter` classes, the letter "D" represents the day in a year. However, the number of occurrences is interpreted differently by the classes.

If the date - July 19, 2023 is formatted using `SimpleDateFormat("D")`, the output results in 231.

If the same date - July 19, 2023 is formatted using `DateTimeFormatter("D")`, the output results in the following error:

```
Exception in thread "main" java.time.DateTimeException: Field DayOfYear cannot be printed as the value 200 exceeds the maximum print width of 2
```

For more details about the syntax and behavior of these UDFs, see [Hive LanguageManual UDF](#).



Note: Starting from CDP Private Cloud Base 7.1.7 SP2 version Cumulative hotfix 14, a new configurable `hive.datetime.formatter` property is introduced through [HIVE-25576](#) that enables you to choose between `SimpleDateFormat` and `DateTimeFormatter` for the `unix_timestamp` and `from_unixtime` SQL functions.

Semantic changes and workarounds CDP 7.1.7 SP2 CHFx

Review the semantic changes in Hive after migrating to CDP 7.1.7 SP2 CHFx from CDP 7.1.7 SP1. A link to Apache Hive JIRAs, if there is one, provides more information about the semantic change. If you are migrating to CDP 7.1.7 SP2 CHFx from CDH 6.2.1, review the list of Hive changes in each CDP release prior to CDP 7.1.7 SP2 CHFx.

New property to control datetime formatter

Learn about the new property that is introduced that allows you to choose the datetime formatter class to be used for formatting and parsing date and timestamp.

Before Upgrade to 7.1.7 SP2 Cumulative hotfix 14

The `unix_timestamp` and `from_unixtime` functions were previously enhanced to use the `DateTimeFormatter` class for formatting and parsing date and timestamp instead of the `SimpleDateFormat` class.

After Upgrade to 7.1.7 SP2 Cumulative hotfix 14

[HIVE-25576](#) introduces a configurable `hive.datetime.formatter` property that you can use to choose between `SimpleDateFormat` and `DateTimeFormatter` for the `unix_timestamp` and `from_unixtime` SQL functions.

The two Java datetime formatters differ in their behavior, which leads to different query results. The supported patterns between the two formatters are also different, which makes existing queries crash during runtime (after an upgrade). Also, adapting to the new behavior of the `DateTimeFormatter` class can be challenging and time-consuming for users considering that the `unixtime` functions are extensively used.

Although the `DateTimeFormatter` class is an improvement over `SimpleDateFormat`, some users may want to retain the old behavior to ensure compatibility after migration, therefore, making it necessary for introducing this property.

The possible values for the `hive.datetime.formatter` property are 'DATETIME' and 'SIMPLE' representing `DateTimeFormatter` and `SimpleDateFormat` respectively. The default value is set to 'DATETIME'.

Action Required

Set the `hive.datetime.formatter` parameter to 'SIMPLE' if you want to use the `SimpleDateFormat` class. Add this parameter in Cloudera Manager (Hive Service Advanced Configuration Snippet (Safety Valve) for `hive-site.xml`).

Dates are parsed by ignoring trailing invalid characters

Learn about the change in the way dates are parsed from a string by ignoring trailing invalid characters

Before upgrade to CDP 7.1.7 SP2 Cumulative hotfix 14

[HIVE-20007](#) introduced changes in the way dates were parsed from strings. SQL functions or date operations involving invalid dates returned "null".

After upgrade to CDP 7.1.7 SP2 Cumulative hotfix 14

[HIVE-27586](#) extracts and returns a valid date from a string value if there is a valid date prefix in the string. This fix partially restores the behavior changes introduced as part of HIVE-20007 and also makes the current behavior of handling trailing invalid characters more consistent.

The following table illustrates the behavior changes before and after the fix:

String value	Behavior (before HIVE-20007)	Previous behavior (after HIVE-20007)	Current behavior (after HIVE-27586)
2023-08-03_16:02:00	2023-08-03	null	2023-08-03
2023-08-03-16:02:00	2023-08-03	null	2023-08-03
2023-08-0316:02:00	2024-06-11	null	2023-08-03
03-08-2023	0009-02-12	null	0003-08-20
2023-08-03 GARBAGE	2023-08-03	2023-08-03	2023-08-03
2023-08-03TGARBAGE	2023-08-03	2023-08-03	2023-08-03
2023-08-03_GARBAGE	2023-08-03	null	2023-08-03

This change affects various Hive SQL functions and operators that accept dates from string values, such as CAST (V AS DATE), CAST (V AS TIMESTAMP), TO_DATE, DATE_ADD, DATE_DIFF, WEEKOFYEAR, DAYOFWEEK, and TRUNC.

Semantic changes and workarounds CDP 7.1.8 CHFx

Review the semantic changes in Hive after migrating to CDP 7.1.8 CHFx from CDP 7.1.7 SP2. A link to Apache Hive JIRAs, if there is one, provides more information about the semantic change. If you are migrating to CDP 7.1.8 CHFx from CDH 6.2.1, review the list of Hive changes in each CDP release prior to CDP 7.1.8 CHFx.

Handling table column named default

Learn about the change to ensure that Hive is able to reference a table column that is named as 'default'.

Before upgrade to CDP 7.1.8

Hive was unable to reference table columns that are named as 'default' and returns NULL. For example,

```
create table t1 (a int, `default` int) stored as orc TBLPROPERTIES ('transactional'='true');
insert into t1 values (1, 2), (10, 11);
update t1 set a = `default`;
select * from t1;
Output:
NULL    NULL
NULL    NULL

Expected output:
2        2
11       11
```

After upgrade to 7.1.8

[HIVE-25969](#) fixes this issue and Hive is now able to reference table columns named 'default'.

Fix precision and scale inference for aggregate rewriting in Calcite

This change fixes an issue where type inference of intermediate precision and scale for division is not correct.

Before upgrade to CDP 7.1.8 CHF2

The `AggregateReduceFunctionsRule` class of Calcite rules reduce aggregate functions into simpler forms, for example, `avg(x)` into `sum(x)/count(x)`. When the type of `avg(x)` aggregate function is decimal, type inference of the intermediate precision and scale for the division is not done correctly.

The reason is due to lack of support for some types in the `getDefaultPrecision` method in `HiveTypeSystemImpl`.

After upgrade to CDP 7.1.8 CHF2

[HIVE-22978](#) provides the fix to correct the precision and scale type inference for aggregate rewriting in Calcite. Additionally, the `deriveSumType` method in `HiveTypeSystemImpl` is overridden to abide by the Hive semantics for sum aggregate type inference.

Migrating Spark Apps

Preventing SparkSQL incompatibility

You need to be aware of two SparkSQL incompatibilities and how to work around these problems. The upgrade process converted all CDH Hive tables to external tables, however, if you moved managed, non-ACID tables preventing conversion to external tables, these are not compatible with native SparkSQL. Also, you might encounter a problem reading Hive 2 external ORC tables from Spark.

Related Information

[Hive Warehouse Connector for accessing Apache Spark data](#)

[Hive-17275](#)

[SPARK-28098](#)

Managed, non-ACID table problem

About this task

Consider using either one of the following options:

- Convert ACID tables to external tables after the Cloudera upgrade.
- Use the Hive Warehouse Connector.

Follow these steps to create a new external table using Hive 3 and migrate the data from the managed table to the new table.

Procedure

1. Run the `SHOW CREATE TABLE` statement on the original table to get the full definition of the table.

```
SHOW CREATE TABLE <tablename>;
```

2. Rename the managed table to `*_old`.
3. Migrate data from `*_old` to `<new>` external table using the original name in the historical, or the default, location (`/warehouse/tablespace/external/hive/<?>.db/<tablename>`).

```
CREATE EXTERNAL TABLE new_t AS SELECT * FROM old_t;
```

Reading a Hive external table in ORC from Spark

About this task

Unlike the Hive ORC reader, which supports recursive directory reads, the Spark native ORC reader does not support recursive directory reads of Hive tables. Reading Hive 2.x tables is a problem under both of the following circumstances:

- You created the table using Hive CTAS (create table as select).
- One of more selected tables included UNION ALL.

When creating a table under these circumstances, subdirectories are named /1 /2 /3. Subdirectories do not include the HIVE_UNION_SUBDIR_ prefix as Hive 3-created tables do. You cannot read these tables from Spark if the tables are in ORC format.

The following workaround configures Spark to overcome this problem.

Procedure

1. If you already started the Spark shell, quit the shell.
You cannot perform workaround configuration for the session if the session is already started.
2. Start the Spark shell with convertMetastoreOrc disabled.
For example:

```
spark-shell ...  
--conf spark.sql.hive.convertMetastoreOrc=false
```

Spark integration with Hive

You need to know a little about Hive Warehouse Connector (HWC) and how to find more information because to access Hive from Spark, you need to use HWC implicitly or explicitly.

You can use the Hive Warehouse Connector (HWC) to access Hive managed tables from Spark. HWC is specifically designed to access managed ACID v2 Hive tables, and supports writing to tables in Parquet, ORC, Avro, or Textfile formats. HWC is a Spark library/plugin that is launched with the Spark app.

You do not need HWC to read from or write to Hive external tables. Spark uses native Spark to access external tables.

Use the Spark Direct Reader and HWC for ETL jobs. For other jobs, consider using Apache Ranger and the HiveWarehouseConnector library to provide row and column, fine-grained access to the data.

HWC supports spark-submit and pyspark. The spark thrift server is not supported.

Related Information

[Hive Warehouse Connector for accessing Apache Spark data](#)

Removing Hive on Spark Configurations

Your scripts, or queries, include the Hive on Spark configuration, which is no longer supported, and you must know how to recognize and remove these configurations.

In Cloudera, there is no Hive-Spark dependency. The Spark site and libs are not in the classpath. This execution engine has been replaced by Apache Tez.

Before Upgrade to Cloudera

CDH supported Hive on Spark and the following configuration to enable Hive on Spark: set hive.execution.engine=spark

After Upgrade to Cloudera

Cloudera does not support Hive on Spark. Scripts that enable Hive on Spark do not work.

Action Required

Remove set hive.execution.engine=spark from your scripts.

Disabling Partition Type Checking

An enhancement in Hive 3 checks the types of partitions. This feature can be disabled by setting a property. For more information, see the ASF Apache Hive Language Manual.

Before Upgrade to Cloudera

In CDH 5.x, partition values are not type checked.

After Upgrade to Cloudera

Partition values specified in the partition specification are type checked, converted, and normalized to conform to their column types if the property `hive.typecheck.on.insert` is set to true (default). The values can be numbers.

Action Required

If type checking of partitions causes problems, disable the feature. To disable partition type checking, set `hive.typecheck.on.insert` to false. For example:

```
SET hive.typecheck.on.insert=false;
```

Related Information

[Apache Hive Wiki: Partitioned Tables](#)

[Hive Language Manual: Alter Partition](#)

Converting Hive CLI scripts to Beeline

If you have legacy scripts that run Hive queries from edge nodes using the Hive CLI, you must solve potential incompatibilities with variable substitution in these scripts. CDP 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 allowlist (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 run 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. Include these configuration parameters in the allowlist: In Cloudera Manager, go to Clusters HIVE_ON_TEZ-1 Configuration, and search for hive-site.
7. In HiveServer2 Advanced Configuration Snippet (Safety Valve) for hive-site.xml, add the property key: hive.security.authorization.sqlstd.confwhitelist.append.
8. Provide the property value, or values, to allowlist, for example: mylocal\.*|junk.
This action appends mylocal.test.var and mylocal.test.env.var parameters to the allowlist.
9. Save configuration changes, and restart any components as required.
10. Run a command that references HIVEVAR to parse a SQL script, removes the incompatible env namespace, and runs 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 |
+-----+-----+
```

Hive unsupported interfaces and features

You need to understand the interfaces that are not supported.

Unsupported Interfaces and features

The following interfaces are not supported in Cloudera Base on premises:

- Druid
- Hcat CLI (however HCatalog is supported)
- Hive CLI (replaced by Beeline)
- Hive View UI feature in Ambari
- Apache Hive Standalone driver
- Renaming Hive databases

- Multiple insert overwrite queries that read data from a source table.
 - LLAP
 - MapReduce execution engine (replaced by Tez)
 - Pig
 - S3 for storing tables (available in Cloudera on cloud only)
 - Spark execution engine (replaced by Tez)
 - Spark thrift server
- Spark and Hive tables interoperate using the Hive Warehouse Connector.
- SQL Standard Authorization
 - Storage Based Authorization
 - Tez View UI feature in Ambari
 - WebHCat

You can use Hue in lieu of Hive View.

Storage Based Authorization

Storage Based Authorization (SBA) is no longer supported in Cloudera. Ranger integration with Hive metastore provides consistency in Ranger authorization enabled in HiveServer (HS2). SBA did not provide authorization support for metadata that does not have a file/directory associated with it. Ranger-based authorization has no such limitation.

Hive-Kudu integration

Cloudera does not support the integration of HiveServer (HS2) with Kudu tables. You cannot run queries against Kudu tables from HS2.

Partially unsupported interfaces

Apache Hadoop Distributed Copy (DistCP) is not supported for copying Hive ACID tables.

Unsupported Features

Cloudera does not support the following features that were available in HDP and CDH platforms:

- CREATE TABLE that specifies a managed table location
- Do not use the LOCATION clause to create a managed table. Hive assigns a default location in the warehouse to managed tables. That default location is configured in Hive using the `hive.metastore.warehouse.dir` configuration property, but can be overridden for the database by setting the `CREATE DATABASE MANAGEDLOCATION` parameter.
- CREATE INDEX and related index commands were removed in Hive 3, and consequently are not supported in Cloudera.
- In Cloudera, you use the Hive 3 default ORC columnar file formats to achieve the performance benefits of indexing. Materialized Views with automatic query rewriting also improves performance. Indexes migrated to Cloudera are preserved but render any Hive tables with an undroppable index. To drop the index, google the Known Issue for CDPD-23041.
- Hive metastore (HMS) high availability (HA) load balancing in CDH
- You need to set up HMS HA as described in the documentation.
- Local or Embedded Hive metastore server
- Cloudera does not support the use of a local or embedded Hive metastore setup.

Unsupported Connector Use

Cloudera does not support the Sqoop exports using the Hadoop jar command (the Java API) that Teradata documents. For more information, see [Migrating data using Sqoop](#).

Migrating Hive Workloads from HDP 2.6.5 after an in-place upgrade

You upgraded from HDP 2.6.5 to Cloudera Private Cloud Base. The upgrade moved the Hive data and schema to Cloudera Private Cloud Base. As the Hive Administrator, you need to make Hive tables available to your users. You need to configure your Hive-related services for CDP, and secure access to Hive data.

Assumptions

- You are familiar with Apache Hive 3.1 key features and supported interfaces.
- You acquired basic information about the CDP platform before you upgraded from HDP.

Related Information

[Apache Hive 3 Key Features](#)

[Apache Hive 3 Architectural Overview](#)

Changes to HDP Hive tables

As a Data Scientist, Architect, Analyst, or other Hive user you need to locate and use your Apache Hive 3 tables after an upgrade. You also need to understand the changes that occur during the upgrade process.

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 upgrade process from HDP to CDP Private Cloud Base moves managed files to the Hive warehouse `/warehouse/tablespace/managed/hive`. The upgrade process carries the external files over to CDP Private Cloud Base with no change in location. By default, Hive places any new external tables you create in `/warehouse/tablespace/external/hive`. The upgrade process sets the `hive.metastore.warehouse.dir` property to this location, designating it the Hive warehouse location.

Changes to table references using dot notation

Upgrading to Cloudera includes the Hive-16907 bug fix, which rejects ``db.table`` in SQL queries. The dot (.) is not allowed in table names. To reference the database and table in a table name, both must be enclosed in backticks as follows: ``db`.`table``.

Changes to ACID properties

Hive 3.x in Cloudera Base on premises supports 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 was ACID v1. In Hive 3.x, the mature version of ACID is ACID v2, which is the default table type in CDP Private Cloud Base.

Native and non-native storage formats

Storage formats are a factor in upgrade changes to table types. Hive 2.x and 3.x support the following 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

CDP Private Cloud Base upgrade changes to HDP table types

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

Table 2: HDP 2.x and Cloudera Table Type Comparison

HDP 2.x				CDP	
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
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

Checking and correcting Hive table locations

As a Data Engineer, you need to understand the relocation of files after the upgrade process. The file type and other factors affect the relocation during the upgrade.

About this task

The upgrade process changes table types in some cases. The following table compares Hive table types and ACID operations before and after upgrading. The ownership of the Hive table file is a factor in determining table types and ACID operations after the upgrade.

Table 4.1. Before and After Upgrading Table Type Comparison

Before Upgrading	After Upgrading
------------------	-----------------

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
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](#).

** Not SparkSQL-compatible

If you had external files before the upgrade, the upgrade process carries the external files over to CDP after upgrading with no change in location. The external files continue to reside in the /apps/hive/warehouse directory.

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 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 directory is in a different encryption zone than the new warehouse directory.

The /apps/hive/warehouse directory, which is the location of the Hive 2.x warehouse before upgrading, might or might not exist after upgrading.

Procedure

1. Check the /apps/hive/warehouse directory for files that do not belong there after upgrading.

Files that do not belong in /apps/hive/warehouse are files described in the table above as managed files after upgrading. The upgrade process should have moved the managed files to the new /warehouse/tablespace/managed/hive/warehouse directory.

2. Check that the upgrade process moved managed files to /warehouse/tablespace/managed/hive.
3. Check that Hive places any new external tables you create after upgrading in /warehouse/tablespace/external/hive.

Configuration changes

Hive Configuration Property Changes

You need to know the property value changes made by the upgrade process as the change might impact your work. You might need to consider reconfiguring property value defaults that the upgrade changes.

Hive Configuration Property Values

The upgrade process changes the default values of some Hive configuration properties and adds new properties. The following list describes those changes that occur after upgrading from CDH or HDP to Cloudera.

datanucleus.connectionPool.maxPoolSize

Before upgrade: 30

After upgrade: 10

datanucleus.connectionPoolingType

Before upgrade: BONECP

After upgrade: HikariCP

hive.auto.convert.join.noconditionaltask.size

Before upgrade: 20971520

After upgrade: 52428800

Exception: Preserves pre-upgrade value if old default is overridden; otherwise, uses new default.

hive.auto.convert.sortmerge.join

Before upgrade: FALSE in the old CDH; TRUE in the old HDP.

After upgrade: TRUE

hive.auto.convert.sortmerge.join.to.mapjoin

Before upgrade: FALSE

After upgrade: TRUE

hive.cbo.enable

Before upgrade: FALSE

After upgrade: TRUE

hive.cbo.show.warnings

Before upgrade: FALSE

After upgrade: TRUE

hive.compactor.worker.threads

Before upgrade: 0

After upgrade: 5

hive.compute.query.using.stats

Before upgrade: FALSE

After upgrade: TRUE

hive.conf.hidden.list

Before upgrade:

```
javax.jdo.option.ConnectionPassword,hive.server2.keystore.password,hive.metastore.dbaccess.ssl.truststore.password,fs.s3.awsAccessKeyId,fs.s3.awsSecretAccessKey,fs.s3n.awsAccessKeyId,fs.s3n.awsSecretAccessKey,fs.s3a.access.key,fs.s3a.secret.key,fs.s3a.proxy.password,dfs.adls.oauth2.credential,fs.adl.oauth2.credential,fs.azure.account.oauth2.client.secret
```

After upgrade:

```
javax.jdo.option.ConnectionPassword,hive.server2.keystore.password,hive.druid.metadata.password,hive.driver.parallel.compilation.global.limit
```

hive.conf.restricted.list

Before upgrade:

```
hive.security.authenticator.manager,hive.security.authorization.
manager,hive.users.in.admin.role,hive.server2.xsrf.filter.enable
d,hive.spark.client.connect.timeout,hive.spark.client.server.con
nect.timeout,hive.spark.client.channel.log.level,hive.spark.clie
nt.rpc.max.size,hive.spark.client.rpc.threads,hive.spark.client.
secret.bits,hive.spark.client.rpc.server.address,hive.spark.clie
nt.rpc.server.port,hive.spark.client.rpc.sasl.mechanisms,hadoop.
bin.path,yarn.bin.path,spark.home,bonecp.,hikaricp.,hive.driver.
parallel.compilation.global.limit,_hive.local.session.path,_hive
.hdfs.session.path,_hive.tmp_table_space,_hive.local.session.pat
h,_hive.hdfs.session.path,_hive.tmp_table_space
```

After upgrade:

```
hive.security.authenticator.manager,hive.security.authorization.
manager,hive.security.metastore.authorization.manager,hive.secur
ity.metastore.authenticator.manager,hive.users.in.admin.role,hiv
e.server2.xsrf.filter.enabled,hive.security.authorization.enable
d,hive.distcp.privileged.doAs,hive.server2.authentication.ldap.b
aseDN,hive.server2.authentication.ldap.url,hive.server2.authenti
cation.ldap.Domain,hive.server2.authentication.ldap.groupDNPatte
rn,hive.server2.authentication.ldap.groupFilter,hive.server2.aut
hentication.ldap.userDNPattern,hive.server2.authentication.ldap.
userFilter,hive.server2.authentication.ldap.groupMembershipKey,h
ive.server2.authentication.ldap.userMembershipKey,hive.server2.a
uthentication.ldap.groupClassKey,hive.server2.authentication.lda
p.customLDAPQuery,hive.privilege.synchronizer.interval,hive.spar
k.client.connect.timeout,hive.spark.client.server.connect.timeou
t,hive.spark.client.channel.log.level,hive.spark.client.rpc.max.
size,hive.spark.client.rpc.threads,hive.spark.client.secret.bits
,hive.spark.client.rpc.server.address,hive.spark.client.rpc.serv
er.port,hive.spark.client.rpc.sasl.mechanisms,bonecp.,hive.druid
.broker.address.default,hive.druid.coordinator.address.default,h
ikaricp.,hadoop.bin.path,yarn.bin.path,spark.home,hive.driver.pa
rallel.compilation.global.limit,_hive.local.session.path,_hive.h
dfs.session.path,_hive.tmp_table_space,_hive.local.session.path,
_hive.hdfs.session.path,_hive.tmp_table_space
```

hive.default.fileformat.managed

Before upgrade: None

After upgrade: ORC

hive.default.rcfile.serde

Before upgrade: org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe

After upgrade: org.apache.hadoop.hive.serde2.columnar.LazyBinaryColumnarSerDe

Not supported in Impala. Impala cannot read Hive-created RC tables.

hive.driver.parallel.compilation

Before upgrade: FALSE

After upgrade: TRUE

hive.exec.dynamic.partition.mode

Before upgrade: strict

After upgrade: nonstrict

In Cloudera Base on premises, accidental use of dynamic partitioning feature is not prevented by default.

hive.exec.max.dynamic.partitions

Before upgrade: 1000

After upgrade: 5000

In Cloudera Base on premises, fewer restrictions on dynamic partitioning occur than in the pre-upgrade CDH or HDP cluster.

hive.exec.max.dynamic.partitions.pernode

Before upgrade: 100

After upgrade: 2000

In Cloudera Base on premises, fewer restrictions on dynamic partitioning occur than in the pre-upgrade CDH or HDP cluster.

hive.exec.post.hooks

Before upgrade:

```
com.cloudera.navigator.audit.hive.HiveExecHookContext,org.apache.hadoop.hive ql.hooks.LineageLogger
```

After upgrade: org.apache.hadoop.hive.ql.hooks.HiveProtoLoggingHook

A prime number is recommended.

hive.exec.reducers.max

Before upgrade: 1099

After upgrade: 1009

Exception: Preserves pre-upgrade value if old default is overridden; otherwise, uses new default

hive.execution.engine

Before upgrade: mr

After upgrade: tez

Tez is now the only supported execution engine, existing queries that change execution mode to Spark or MapReduce within a session, for example, fail.

hive.fetch.task.conversion

Before upgrade: minimal

After upgrade: more

hive.fetch.task.conversion.threshold

Before upgrade: 256MB

After upgrade: 1GB

Exception: Preserves pre-upgrade value if old default is overridden; otherwise, uses new default.

hive.hashtable.key.count.adjustment

Before upgrade: 1

After upgrade: 0.99

Exception: Preserves pre-upgrade value if old default is overridden; otherwise, uses new default.

hive.limit.optimize.enable

Before upgrade: FALSE

After upgrade: TRUE

hive.limit.pushdown.memory.usage

Before upgrade: 0.1

After upgrade: 0.04

Exception: Preserves pre-upgrade value if old default is overridden; otherwise, uses new default.

hive.mapjoin.hybridgrace.hashtable

Before upgrade: TRUE

After upgrade: FALSE

hive.mapred.reduce.tasks.speculative.execution

Before upgrade: TRUE

After upgrade: FALSE

hive.metastore.aggregate.stats.cache.enabled

Before upgrade: TRUE

After upgrade: FALSE

hive.metastore.disallow.incompatible.col.type.changes

Before upgrade: FALSE

After upgrade: TRUE

Schema evolution is more restrictive in Cloudera Base on premises than in CDH to avoid data corruption. The new default disallows column type changes if the old and new types are incompatible.

hive.metastore.dml.events

Before upgrade: FALSE

After upgrade: TRUE

hive.metastore.event.message.factory

Before upgrade: org.apache.hadoop.hive.metastore.messaging.json.ExtendedJSONMessageFactory

After upgrade: org.apache.hadoop.hive.metastore.messaging.json.gzip.GzipJSONMessageEncoder

hive.metastore.uri.selection

Before upgrade: SEQUENTIAL

After upgrade: RANDOM

hive.metastore.warehouse.dir

Before upgrade from CDH: /user/hive/warehouse

Before upgrade from HDP: /apps/hive/warehouse

After upgrade from CDH: /warehouse/tablespace/managed/hive

After upgrade from HDP: /warehouse/tablespace/managed/hive

For information about the location of old tables and new tables, which you create after the upgrade, see [Changes to CDH Hive Tables](#) or [Changes to HDP Hive tables](#).

hive.optimize.metadataonly

Before upgrade: FALSE

After upgrade: TRUE

hive.optimize.point.lookup.min

Before upgrade: 31

After upgrade: 2

hive.prewarm.numcontainers

Before upgrade: 10

After upgrade: 3

hive.script.operator.env.blacklist

Before upgrade: hive.txn.valid.txns,hive.script.operator.env.blacklist

After upgrade: hive.txn.valid.txns,hive.txn.tables.valid.writeids,hive.txn.valid.writeids,hive.script.operator.env.blacklist

hive.security.authorization.sqlstd.confwhitelist

Before upgrade:

```
hive\auto\.*hive\cbo\.*hive\convert\.*hive\exec\dynamic\
.partition.*hive\exec\.*\dynamic\partitions\.*hive\exec\c
ompress\.*hive\exec\infer\.*hive\exec\mode.local\.*hive\
exec\orc\.*hive\exec\parallel.*hive\explain\.*hive\fetch.
task\.*hive\groupby\.*hive\hbase\.*hive\index\.*hive\ind
ex\.*hive\intermediate\.*hive\join\.*hive\limit\.*hive\l
og\.*hive\mapjoin\.*hive\merge\.*hive\optimize\.*hive\or
c\.*hive\outerjoin\.*hive\parquet\.*hive\ppd\.*hive\prew
arm\.*hive\server2\proxy\userhive\skewjoin\.*hive\smbjoin
\.*hive\stats\.*hive\strict\.*hive\tez\.*hive\vectorized
\.*mapred\map\.*mapred\reduce\.*mapred\output\compression
\codecmapped\job\queuenamemapred\output\compression\typema
pred\min\split\sizemapreduce\job\reduce\slowstart\complet
edmapsmapped\job\queuenamemapreduce\job\tagmapreduce\in
put\fileinputformat\split\minsizemapreduce\map\.*mapreduce\
.reduce\.*mapreduce\output\fileoutputformat\compress\codecm
apreduce\output\fileoutputformat\compress\typeoozie\.*tez\
am\.*tez.task\.*tez.runtime\.*tez.queue.namehive.transpo
se\aggr\joinhive\exec\reducers.bytes.per.reducerhive\cli
ent\stats\countershive\exec\default\partition.namehive\ex
ec\drop\ignorenonexistenthive\counters.group.namehive\defa
ult\fileformat.managedhive\enforce.bucketmapjoinhive\enforc
e.sortmergebucketmapjoinhive.cache.expr.evaluationhive\quer
y.result\fileformathive hashtable.loadfactorhive hashtable\
.initialCapacityhive.ignore.mapjoin.hinhive.limit.row.max
\sizehive.mapred.modehive.map.aggrhive.compute.query.usi
ng.statshive\exec.rowoffsethive.variable.substitutehive.va
riable.substitute.depthhive.autogen.columnalias.prefix.inc
ludefuncnamehive.autogen.columnalias.prefix.labelhive\exec\
.check.crossproductshive.cli.tez.session.asynhive.compath
ive\exec.concatenate.check.indexhive.display.partition.co
ls.separatelyhive.error.on.empty.partitionhive.execution\
enginehive\exec.copyfile.maxsizehive.exim.uri.scheme.whit
elisthive.file.max.footerhive.insert.into.multilevel.dirs
hive.localize.resource.num.wait.attemptshive.multi.insert
.move.tasks.share.dependencieshive.support.quoted.identif
iershive.resultset.use.unique.column.nameshive.analyze.st
mt.collect.partlevel.statshive\exec.schema.evolutionhive\
server2.logging.operation.levelhive.server2.thrift.results
et.serialize.in.taskshive.support.special.characters.tabl
enamehive\exec.job.debug.capture.stacktraceshive\exec.job
.debug.timeouthive.llap.io.enabledhive.llap.io.use.file
id.pathhive.llap.daemon.service.hosthive.llap.execution\
.modehive.llap.auto.allow.uberhive.llap.auto.enforce.tre
ehive.llap.auto.enforce.vectorizedhive.llap.auto.enforce\
.statshive.llap.auto.max.input.sizehive.llap.auto.max.o
utput.sizehive.llap.skip.compile.udf.checkhive.llap.clie
nt.consistent.splitshive.llap.enable.grace.join.in.llaph
ive.llap.allow.permanent.fnshive\exec.max.created.filessh
ive\exec.reducers.maxhive.reorder.nway.joinhive.output\.
```

```
file\extensionhive\exec\show\job\failure\debug\infohive\
exec\tasklog\debug\timeouthive\query\id
```

After upgrade:

```
hive\auto\.*hive\cbo\.*hive\convert\.*hive\druid\.*hive\
exec\dynamic\partition.*hive\exec\max\dynamic\partitions.
.*hive\exec\compress\.*hive\exec\infer\.*hive\exec\mode.l
ocal\.*hive\exec\orc\.*hive\exec\parallel.*hive\exec\que
ry\redactor\.*hive\explain\.*hive\fetch.task\.*hive\group
by\.*hive\hbase\.*hive\index\.*hive\index\.*hive\interme
diate\.*hive\jdbc\.*hive\join\.*hive\limit\.*hive\log\.*
hive\mapjoin\.*hive\merge\.*hive\optimize\.*hive\materia
lizedview\.*hive\orc\.*hive\outerjoin\.*hive\parquet\.*hi
ve\ppd\.*hive\prewarm\.*hive\query\redaction\.*hive\serv
er2\thrift\resultset\default\fetch\sizehive\server2\proxy
\userhive\skewjoin\.*hive\smbjoin\.*hive\stats\.*hive\st
rict\.*hive\tez\.*hive\vectorized\.*hive\query\reexecutio
n\.*reexec\overlay\.*fs\defaultFSssl\client\truststore\lo
cationdistcp\atomicdistcp\ignore\failuresdistcp\preserve\st
atusdistcp\preserve\rawattrsdistcp\sync\foldersdistcp\dele
te\missing\sourcedistcp\keystore\resourcedistcp\liststatus\
.threadsdistcp\max\mapsdistcp\copy\strategydistcp\skip\crc
distcp\copy\overwritdistcp\copy\appenddistcp\map\bandwidt
h\mbdistcp\dynamic\.*distcp\meta\folderdistcp\copy\listin
g\classdistcp\filters\classdistcp\options\skipcrccheckdistc
p\options\mdistcp\options\numListstatusThreadsdistcp\option
s\mapredSslConfdistcp\options\bandwidthdistcp\options\overw
ritdistcp\options\strategydistcp\options\idistcp\options\.
p.*distcp\options\updatedistcp\options\deletemapred\map\.*
mapred\reduce\.*mapred\output\compression\codecmapred\job\
.queue\namemapred\output\compression\typemapred\min\split\
.sizemapreduce\job\reduce\slowstart\completedmapsmareduce\
.job\queuenamemapreduce\job\tagmapreduce\input\fileinputfor
mat\split\minsizemapreduce\map\.*mapreduce\reduce\.*mapred
uce\output\fileoutputformat\compress\codecmapreduce\output\
.fileoutputformat\compress\typeoozie\.*tez\am\.*tez\task\.
.*tez\runtime\.*tez\queue\namehive\transpose\aggr\joinhiv
e\exec\reducers\bytes\per\reducerhive\client\stats\count
erhive\exec\default\partition.namehive\exec\drop\ignoren
onexistenthive\counters\group.namehive\default\fileformat\
managedhive\enforce\bucketmapjoinhive\enforce\sortmergebucke
tmapjoinhive\cache\expr\evaluationhive\query\result\filefo
rmathive\hashtable\loadfactorhive\hashtable\initialCapacityh
ive\ignore\mapjoin\hinhive\limit\row\max\sizehive\mapre
d\modehive\map\aggrhive\compute\query\using\statshive\ex
ec\rowoffsethive\variable\substitutehive\variable\substitut
e\depthhive\autogen\columnalias\prefix\includefuncnamehive\
.autogen\columnalias\prefix\labelhive\exec\check\crossprod
uctshive\cli\tez\session\asynhive\compathive\display\par
tition\cols\separatelyhive\error\on\empty\partitionhive\ex
ecution\enginehive\exec\copyfile\maxsizehive\exim\uri\sc
heme\whitelisthive\file\max\footerhive\insert\into\multil
evel\dirshive\localize\resource\num\wait\attemptshive\mul
ti\insert\move\tasks\share\dependencieshive\query\results
\cache\enabledhive\query\results\cache\wait\for\pending\
.resultshive\support\quoted\identifiershive\resultset\use\
.unique\column\namehive\analyze\stmt\collect\partlevel\st
atshive\exec\schema\evolutionhive\server2\logging\operatio
n\levelhive\server2\thrift\resultset\serialize\in\taskshiv
e\support\special\characters\tablenamehive\exec\job\debu
g\capture\stacktraceshive\exec\job\debug\timeouthive\llap
\io\enabledhive\llap\io\use\fileid\pathhive\llap\daemon
```

```
\.service\.hostshive\.llap\.execution\.modehive\.llap\.auto\.allow\.uberhive\.llap\.auto\.enforce\.treehive\.llap\.auto\.enforce\.vectorizedhive\.llap\.auto\.enforce\.statshive\.llap\.auto\.max\.input\.sizehive\.llap\.auto\.max\.output\.sizehive\.llap\.skip\.compile\.udf\.checkhive\.llap\.client\.consistent\.splitshive\.llap\.enable\.grace\.join\.in\.llaphive\.llap\.allow\.permanent\.fnshive\.exec\.max\.created\.fileshive\.exec\.reducers\.maxhive\.reorder\.nway\.joinshive\.output\.file\.extensionhive\.exec\.show\.job\.failure\.debug\.infohive\.exec\.tasklog\.debug\.timeouthive\.query\.idhive\.query\.tag
```

hive.security.command.whitelist

Before upgrade: set,reset,dfs,add,list,delete,reload,compile

After upgrade: set,reset,dfs,add,list,delete,reload,compile,llap

hive.server2.enable.doAs

Before upgrade: TRUE (in case of an insecure cluster only)

After upgrade: FALSE (in all cases)

Affects only insecure clusters by turning off impersonation. Permission issues are expected to arise for affected clusters.

hive.server2.idle.session.timeout

Before upgrade: 12 hours

After upgrade: 24 hours

Exception: Preserves pre-upgrade value if old default is overridden; otherwise, uses new default.

hive.server2.max.start.attempts

Before upgrade: 30

After upgrade: 5

hive.server2.parallel.ops.in.session

Before upgrade: TRUE

After upgrade: FALSE

A Tez limitation requires disabling this property; otherwise, queries submitted concurrently on a single JDBC connection fail or execute slower.

hive.server2.support.dynamic.service.discovery

Before upgrade: FALSE

After upgrade: TRUE

hive.server2.tez.initialize.default.sessions

Before upgrade: FALSE

After upgrade: TRUE

hive.server2.thrift.max.worker.threads

Before upgrade: 100

After upgrade: 500

Exception: Preserves pre-upgrade value if the old default is overridden; otherwise, uses new default.

hive.server2.thrift.resultset.max.fetch.size

Before upgrade: 1000

After upgrade: 10000

hive.service.metrics.file.location

Before upgrade: /var/log/hive/metrics-hiveserver2/metrics.log

After upgrade: /var/log/hive/metrics-hiveserver2-hiveontez/metrics.log

This location change is due to a service name change.

hive.stats.column.autogather

Before upgrade: FALSE

After upgrade: TRUE

hive.stats.deserialization.factor

Before upgrade: 1

After upgrade: 10

hive.support.special.characters.tablename

Before upgrade: FALSE

After upgrade: TRUE

hive.tez.auto.reducer.parallelism

Before upgrade: FALSE

After upgrade: TRUE

hive.tez.bucket.pruning

Before upgrade: FALSE

After upgrade: TRUE

hive.tez.container.size

Before upgrade: -1

After upgrade: 4096

hive.tez.exec.print.summary

Before upgrade: FALSE

After upgrade: TRUE

hive.txn.manager

Before upgrade: org.apache.hadoop.hive ql.lockmgr.DummyTxnManager

After upgrade: org.apache.hadoop.hive ql.lockmgr.DbTxnManager

hive.vectorized.execution.mapjoin.minmax.enabled

Before upgrade: FALSE

After upgrade: TRUE

hive.vectorized.execution.mapjoin.native.fast.hashtable.enabled

Before upgrade: FALSE

After upgrade: TRUE

hive.vectorized.use.row.serde.deserialize

Before upgrade: FALSE

After upgrade: TRUE

Customizing critical Hive configurations

As Administrator, you need property configuration guidelines. You need to know which properties you need to reconfigure after upgrading. You must understand which the upgrade process carries over from the old cluster to the new cluster.

The Cloudera upgrade process tries to preserve your Hive configuration property overrides. These overrides are the custom values you set to configure Hive in the old CDH or HDP cluster. The upgrade process does not preserve all overrides. For example, a custom value you set for `hive.exec.max.dynamic.partitions.pernode` is preserved. In the case of other properties, for example `hive.cbo.enable`, the upgrade ignores any override and just sets the Cloudera-recommended value.

The upgrade process does not preserve overrides to the configuration values of the following properties that you likely need to reconfigure to meet your needs:

- `hive.conf.hidden.list`
- `hive.conf.restricted.list`
- `hive.exec.post.hooks`
- `hive.script.operator.env.blacklist`
- `hive.security.authorization.sqlstd.confwhitelist`
- `hive.security.command.whitelist`

The Apache Hive Wiki describes these properties. The values of these properties are lists.

The upgrade process ignores your old list and sets a new generic list. For example, the `hive.security.command.whitelist` value is a list of security commands you consider trustworthy and want to keep. Any overrides of this list that you set in the old cluster are not preserved. The new default is probably a shorter (more restrictive) list than the original default you were using in the old cluster. You need to customize this Cloudera to meet your needs.

Check and change each property listed above after upgrading as described in the next topic.

Consider reconfiguring more property values than the six listed above. Even if you did not override the default value in the old cluster, the Cloudera default might have changed in a way that impacts your work.

Related Information

[Hive Configuration Property Changes](#)

[Hive Configuration Requirements and Recommendations](#)

[Apache Hive Wiki: Configuration Properties](#)

Setting Hive Configuration Overrides

You need to know how to configure the critical customizations that the upgrade process does not preserve from your old Hive cluster. Referring to your records about your old configuration, you follow steps to set at least six critical property values.

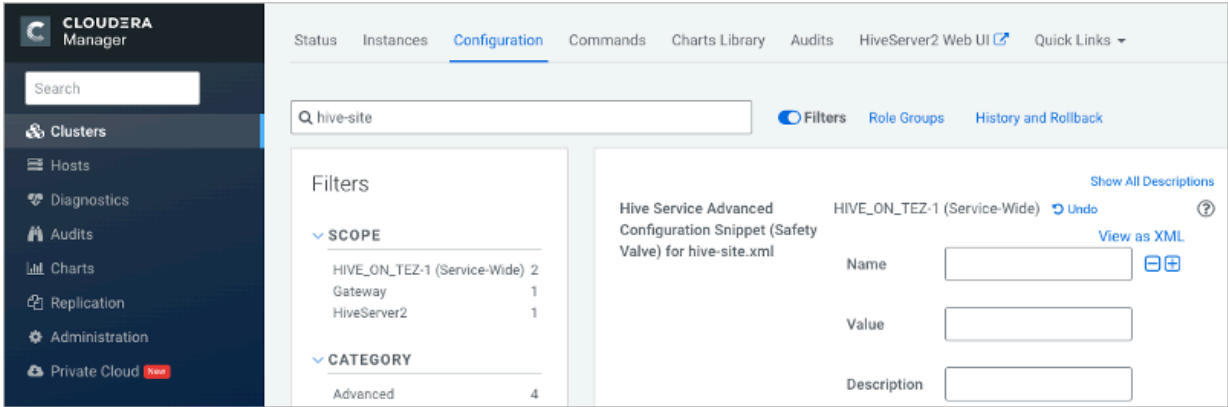
About this task

By design, the six critical properties that you need to customize are not visible in Cloudera Manager, as you can see from the Visible in Cloudera Manager column of Configurations Requirements and Recommendations. You use the Safety Valve to add these properties to `hive-site.xml` as shown in this task.

Procedure

1. In Cloudera Manager Clusters select the Hive on Tez service. Click Configuration, and search for `hive-site.xml`.

2. In Hive Service Advanced Configuration Snippet (Safety Valve) for hive-site.xml, click +.



- 3. In Name, add the hive.conf.hidden.list property.
- 4. In Value, add your custom list.
- 5. Customize the other critical properties: hive.conf.restricted.list, hive.exec.post.hooks, hive.script.operator.env.blacklist, hive.security.authorization.sqlstd.confwhitelist, hive.security.command.whitelist.
Use hive.security.authorization.sqlstd.confwhitelist.append, for example, to set up the list.
- 6. Save the changes and restart the Hive service.
- 7. Look at the Configurations Requirements and Recommendations to understand which overrides were preserved or not.

Related Information
[Hive Configuration Property Changes](#)
[Hive Configuration Requirements and Recommendations](#)
[Apache Hive Wiki: Configuration Properties](#)

Hive Configuration Requirements and Recommendations

You need to set certain Hive and HiveServer (HS2) configuration properties after upgrading. You review recommendations for setting up Cloudera Base on premises for your needs, and understand which configurations remain unchanged after upgrading, which impact performance, and default values.

Requirements and Recommendations

The following table includes the Hive service and HiveServer properties that the upgrade process changes. Other property values (not shown) are carried over unchanged from CDH or HDP to Cloudera

- Set After Upgrade column: properties you need to manually configure after the upgrade to Cloudera. Pre-existing customized values are not preserved after the upgrade.
- Default Recommended column: properties that the upgrade process changes to a new value that you are strongly advised to use.
- Impacts Performance column: properties changed by the upgrade process that you set to tune performance.
- Safety Value Overrides column: How the upgrade process handles Safety Valve overrides.
 - Disregards: the upgrade process removes any old CDH Safety Valve configuration snippets from the new CDP configuration.
 - Preserves means the upgrade process carries over any old CDH snippets to the new CDP configuration.
 - Not applicable means the value of the old parameter is preserved.
- Visible in CM column: property is visible in Cloudera Manager after upgrading. Cloudera Manager after upgrading.

If a property is not visible, and you want to configure it, use the Cloudera Manager Safety Valve to safely add the parameter to the correct file, for example to a cluster-wide, hive-site.xml file.

Table 3:

Property	Set After Upgrade	Default Recommendation	Impacts Performance	New Feature	Safety Valve Overrides	Visible in CM
datanucleus.connectionPool.maxPoolSize			#		Preserve	
datanucleus.connectionPoolingType			#		Disregard	
hive.async.log.enabled					Disregard	#
hive.auto.convert.join.noconditionaltask.size					Not applicable	#
hive.auto.convert.sortmerge.join					Preserve	
hive.auto.convert.sortmerge.join.to.mapjoin					Preserve	
hive.cbo.enable					Disregard	#
hive.cbo.show.warnings					Disregard	
hive.compactor.worker.threads				#	Disregard	#
hive.compute.query.using.stats			#		Disregard	#
hive.conf.hidden.list	#				Disregard	
hive.conf.restricted.list	#				Disregard	
hive.default.fileformat.managed					Disregard	#
hive.default.rcfile.serde		#			Preserve	
hive.driver.parallel.compilation					Disregard	#
hive.exec.dynamic.partition.mode					Disregard	
hive.exec.max.dynamic.partitions					Preserve	
hive.exec.max.dynamic.partitions.pernode					Preserve	
hive.exec.post.hooks	#				Disregard	
hive.exec.reducers.max		# or other prime number			Not applicable	#
hive.execution.engine					Disregard	
hive.fetch.task.conversion			#		Not applicable	#
hive.fetch.task.conversion.threshold			#		Not applicable	#
hive.hashtable.key.count.adjustment			#		Preserve	
hive.limit.optimize.enable		#			Disregard	
hive.limit.pushdown.memory.usage			#		Not Applicable	#
hive.mapjoin.hybridgrace.hashtable		#	#		Disregard	
hive.mapred.reduce.tasks.speculative.execution		#			Disregard	
hive.metastore.aggregate.stats.cache.enabled		#	#		Disregard	
hive.metastore.disallow.incompatible.col.type.changes					Disregard	
hive.metastore.dml.events					Disregard	#
hive.metastore.event.message.factory		#			Disregard	
hive.metastore.uri.selection		#			Disregard	
hive.metastore.warehouse.dir					Preserve	#
hive.optimize.metadataonly		#			Disregard	
hive.optimize.point.lookup.min					Disregard	

Property	Set After Upgrade	Default Recommendation	Impacts Performance	New Feature	Safety Valve Overrides	Visible in CM
hive.prewarm.numcontainers					Disregard	
hive.script.operator.env.blacklist	#				Disregard	
hive.security.authorization.sqlstd.confwhitelist	#				Disregard	
hive.security.command.whitelist	#				Disregard	
hive.server2.enable.doAs					Disregard	#
hive.server2.idle.session.timeout					Not applicable	#
hive.server2.max.start.attempts					Preserve	
hive.server2.parallel.ops.in.session					Preserve	
hive.server2.support.dynamic.service.discovery				#	Disregard	#
hive.server2.tez.initialize.default.sessions				#	Disregard	
hive.server2.thrift.max.worker.threads					Not Applicable	#
hive.server2.thrift.resultset.max.fetch.size					Preserve	
hive.service.metrics.file.location					Disregard	#
hive.stats.column.autogather		#			Disregard	
hive.stats.deserialization.factor		#			Disregard	
hive.support.special.characters.tablename		#			Disregard	
hive.tez.auto.reducer.parallelism				#	Disregard	#
hive.tez.bucket.pruning				#	Disregard	#
hive.tez.container.size				#	Disregard	#
hive.tez.exec.print.summary				#	Disregard	#
hive.txn.manager				#	Disregard	#
hive.vectorized.execution.mapjoin.minmax.enabled		#			Disregard	
hive.vectorized.execution.mapjoin.native.fast.hashtable.enabled		#			Disregard	
hive.vectorized.use.row.serde.deserialize		#			Disregard	

Configuring HMS for high availability

To provide failover to a secondary Hive metastore if your primary instance goes down, you need to know how to add a Metastore role in Cloudera Manager and configure a property.

About this task

Multiple HMS instances run in active/active mode. No load balancing occurs. An HMS client always reaches the first instance unless it is down. In this case, the client scans the `hive.metastore.uris` property that lists the HMS instances for a replacement HMS. The second HMS is the designated replacement if `hive.metastore.uri.selection` is set to `SEQUENTIAL` (recommended and the default); otherwise, the replacement is selected randomly from the list if `hive.metastore.uri.selection` is set to `RANDOM`.

Before you begin

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

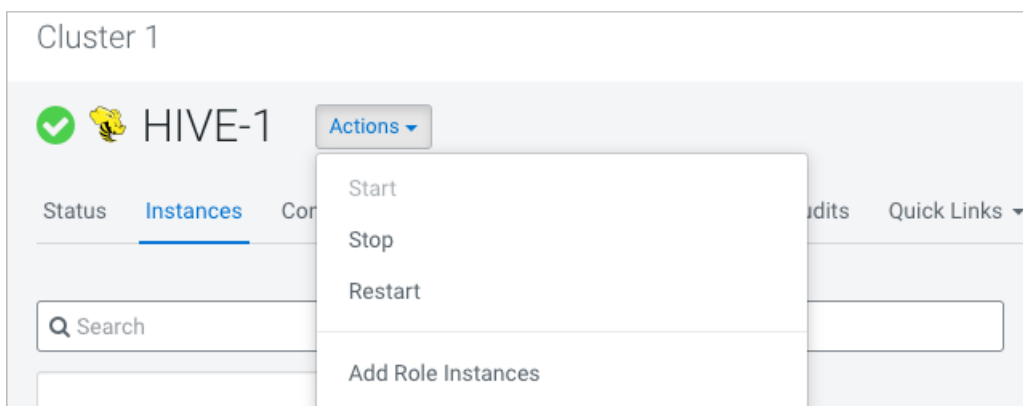
Procedure

1. In Cloudera Manager, click **Clusters** **Hive** **Configuration**.

2. Take one of the following actions:
 - If you have a cluster secured by Kerberos, search for Hive Delegation Token Store, which specifies storage for the Kerberos token as described below.
 - If you have an unsecured cluster, skip the next step.
3. Select `org.apache.hadoop.hive.thrift.DBTokenStore`, and save the change.

Storage for the Kerberos delegation token is defined by the `hive.cluster.delegation.token.store.class` property. The available choices are Zookeeper, the Metastore, and memory. Cloudera recommends using the database by setting the `org.apache.hadoop.hive.thrift.DBTokenStore` property.

4. Click **Instances Actions Add Role Instances**



5. In **Assign Roles**, in **Metastore Server**, click **Select Hosts**.
6. In **Hosts Selected**, scroll and select the host that you want to serve as the backup Metastore, and click **OK**.
7. Click **Continue** until you exit the wizard.
8. Start the Metastore role on the host from the **Actions** menu.
The `hive.metastore.uris` property is updated automatically.
9. To check or to change the `hive.metastore.uri.selection` property, go to **Clusters Hive Configurations**, and search for **Hive Service Advanced Configuration Snippet (Safety Valve) for hive-site.xml**.
10. Add the property and value (**SEQUENTIAL** or **RANDOM**).

Setting up Hive metastore for Atlas

As Administrator, you might plan to recommend Atlas for Hive metadata management and data governance. You have to check that Hive metastore for Atlas is set up, so users can build catalogs of data assets, classify, and govern the assets. If Atlas is not set up you learn how to do so. This section is not applicable if you are upgrading to CDP Private Cloud Base 7.1.7.

About this task

In this task, you set the name of the Atlas service for Hive metastore to use.

Procedure

1. In Cloudera Manager, click **Clusters Hive Configurations**.
2. Search for **Atlas Service**.

3. Choose a method based on the results of your search:

- If Cloudera Manager finds the Atlas Service, check the checkbox to enable the Hive Metastore hook in your Cloudera Manager instance.
- If Cloudera Manager does not find the Atlas Service, in Hive Service Advanced Configuration Snippet (Safety Valve) for atlas-application properties, enter an XML snippet in the value element that provides the name of your Atlas service, myatlasservice in the example below.

```
<property>
  <name>atlas_service</name>
  <value>myatlasservice</value>
</property>
```

4. Save changes.

5. Restart the Hive metastore service.

Changing the Hive warehouse location

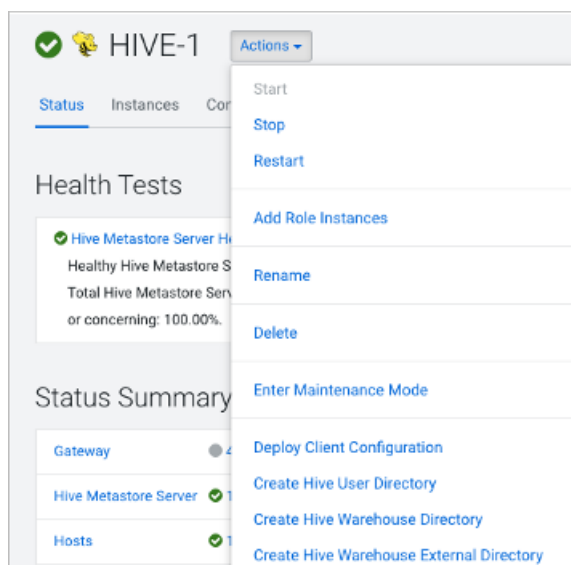
You can change the location of the Hive warehouse by using the configuration settings in your Cloudera Manager instance.

Procedure

1. In Cloudera Manager, click Clusters > Hive (the Hive Metastore service) > Configuration, and change the hive.metastore.warehouse.dir property value to the path you specified for the new Hive warehouse directory.
2. Change the hive.metastore.warehouse.external.dir property value to the path you specified for the Hive warehouse external directory.
3. Save the above configuration changes.

In Cloudera Manager, navigate to the Hive service and from the Actions drop-down, run the services:

- Create Hive Warehouse Directory
- Create Hive Warehouse External Directory



4. Restart the required services for the changes to take effect.

Related Information

[Ranger RMS Authorization for Hive-HDFS](#)

[HDFS ACL Permissions Model](#)

[HDFS ACLS](#)

Removing the LLAP Queue

Before upgrading from HDP to CDP, if you used LLAP, a YARN interactive query queue was created. This queue is carried over to your CDP cluster. You must remove this queue, likely named llap, after the upgrade.


About this task

When you set up LLAP in an HDP cluster, Ambari creates a queue named llap by default; however, you might have created the LLAP queue manually and assigned a different name to the queue. Look for your LLAP and remove it as follows:



Note: LLAP is not supported on Cloudera Private Cloud Base.

Procedure

1. In Cloudera Manager, select Clusters YARN YARN Queue Manager UI .
A graphical queue hierarchy is displayed in the Overview tab.
2. 
Click the options menu for the interactive query queue.
3. Click Delete Queue, and confirm deletion.

Security tasks

After an in-place upgrade to Cloudera, as Administrator, you might need to perform a few security tasks, depending on the type of security you set up, Ranger or HDFS Access Control Lists (ACLs), as well as your data encryption requirements and use of clients to access Hive.

Making the Hive plugin for Ranger visible

After upgrading from HDP or CDH clusters to Cloudera, the Hive plugin for the Hive Metastore and HiveServer2 appears in the Ranger Admin UI unless configuration property problems due to upgrading exist. You can rectify the incorrect properties to fix the problem.

About this task

If the Hive Metastore plugin does not appear in the Ranger Admin UI, you must remove the following property settings from Hive Metastore hive-site.xml safety valve:

- hive.security.authorization.enabled
- hive.security.authorization.manager
- hive.security.metastore.authorization.manager

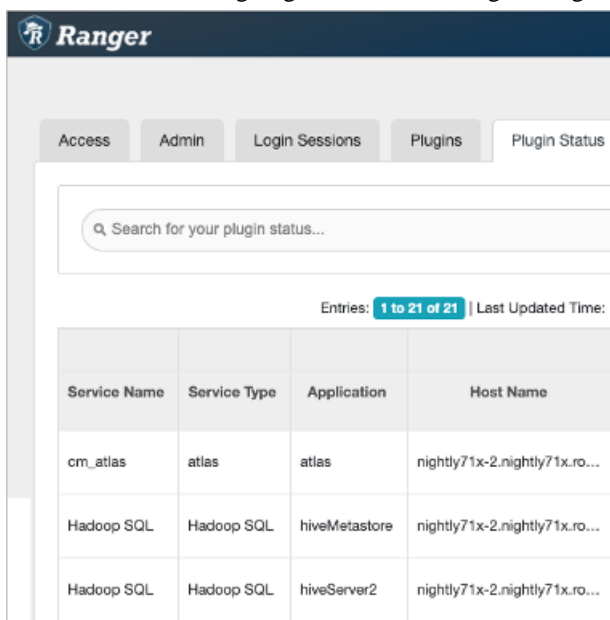
If the HiveServer2 plugin does not appear in the Ranger Admin UI, you must remove the following property settings from HiveServer2 hive-site.xml safety valve:

- hive.security.authorization.enabled
- hive.security.authorization.manager
- hive.security.metastore.authorization.manager
- hive.security.authenticator.manager

After removing these configuration properties, restart the Hive Metastore and HiveServer2 services from Cloudera Manager. Next, you must check whether the Ranger Hive Metastore and HiveServer2 plugins are enabled successfully. To do so:

Procedure

1. From Cloudera Manager, go to Clusters Ranger Ranger Admin Web UI Audit Plugin Status .



Service Name	Service Type	Application	Host Name
cm_atlas	atlas	atlas	nightly71x-2.nightly71x.ro...
Hadoop SQL	Hadoop SQL	hiveMetastore	nightly71x-2.nightly71x.ro...
Hadoop SQL	Hadoop SQL	hiveServer2	nightly71x-2.nightly71x.ro...

The Hadoop SQL service type for the hiveMetastore and hiveServer2 applications should appear. If so, skip the next step. Your configuration is ok.

2. If, after removing the Hive Metastore and HiveServer2 configuration properties from the respective hive-ste.xml safety valves, the Hive Metastore and HiveServer2 plugins are NOT visible, you must confirm whether or not the following configuration properties appear in hive-site.xml:

For Hive Metastore, confirm whether or not the following key-value pair appears in hive-site.xml:

Key: hive.metastore.pre.event.listeners

Value: org.apache.hadoop.hive.ql.security.authorization.plugin.metastore.HiveMetaStoreAuthorizer

If this key-value pair does not appear in hive-site.xml, then add it to the Hive Metastore hive-site.xml safety valve.

For HiveServer2, confirm whether or not the following key value pair appears in hive-site.xml:

Key: hive.security.authenticator.manager

Value: org.apache.hadoop.hive.ql.security.SessionStateUserAuthenticator

If this key-value pair does not appear in hive-site.xml, then add it to the HiveServer2 hive-site.xml safety valve.

Configuring authorization to tables

Although the upgrade process makes no change to the location of external tables, you need to set up access to external tables in HDFS. If you choose the recommended Ranger security model for authorization, you need to set up policies and configure Hive metastore (HMS).

About this task

Set up access to external tables in HDFS using one of the following methods.

- Set up a Hive HDFS policy in Ranger (recommended) to include the paths to external table data.
- Put an HDFS ACL in place. Store the external text file, for example a comma-separated values (CSV) file, in HDFS that will serve as the data source for the external table.

If you want to use Ranger to authorize access to your tables, you must configure a few HMS properties for authorization in addition to setting up Ranger policies. If you have not configured HMS, attempting to create a table using Spark SQL, Beeline, or Hue results in the following error:

```
org.apache.hadoop.hive ql.ddl.DDLTask. MetaException(message:No privilege 'create' found for outputs { database:DATABASE_NAME, table:TABLE_NAME})
```

Related Information

[Authorizing Apache Hive Access](#)

[Configuring HMS properties for authorization](#)

Setting up access control lists

Several sources of information about setting up HDFS ACLs plus a brief Ranger overview and pointer to Ranger information prepare you to set up Hive authorization.

In Cloudera Base on premises, HDFS supports POSIX ACLs (Access Control Lists) to assign permissions to users and groups. In lieu of Ranger policies, you use HDFS ACLs to check and make any necessary changes in HDFS permission changes. For more information, see [HDFS ACLs](#), [Apache Software Foundation HDFS Permissions Guide](#), and [HDFS ACL Permissions](#).

In Ranger, you give multiple groups and users specific permissions based on your use case. You apply permissions to a directory tree instead of dealing with individual files. For more information, see [Authorizing Apache Hive Access](#).

If possible, you should use Ranger policies over HDFS ACLs to control HDFS access. Controlling HDFS access through Ranger provides a single, unified interface for understanding and managing your overall governance framework and policy design. If you need to mimic the legacy Sentry HDFS ACL Sync behavior for Hive and Impala tables, consider using Ranger RMS.

Related Information

[Ranger RMS Authorization for Hive-HDFS](#)

[HDFS ACLS](#)

[Apache Hive 3 Architectural Overview](#)

[Configure a Resource-based Policy: Hive](#)

[Ranger RMS Authorization for Hive-HDFS](#)

[HDFS ACL Permissions Model](#)

Configure encryption zone security

Under certain conditions, you as Administrator, need to perform a security-related task to allow users to access to tables stored in encryption zones. You find out how to prevent access problems to these tables.

About this task

Hive on Tez cannot run some queries on tables stored in encryption zones under certain conditions. Perform the following procedure only when the cluster uses self-signed certificates.



Important: Skip this task for clusters where TLS certificates are properly signed by a Certificate Authority (CA), and the CA is in the truststore files.

Procedure

1. Copy the `ssl-client.xml` file to a directory that is available on all hosts.
2. In Cloudera Manager, click **Clusters Hive on Tez Configuration**.
3. Search for the **Hive Service Advanced Configuration Snippet (Safety Valve) for hive-site.xml** setting.
4. In the **Hive Service Advanced Configuration Snippet (Safety Valve) for hive-site.xml** setting, click **+**.

5. In Name enter the property `tez.aux.uris` and in value enter `path-to-ssl-client.xml`.
Ensure that you include the file URI scheme in the path. For example:

```
tez.aux.uris=file:///etc/hadoof/conf
```

Configure edge nodes as gateways

If you use command-line clients, such as Sqoop, to access Hive, you must configure these gateways to use defaults for your service. You can accomplish this task in a few steps.

About this task

By default, the HS2 instances configured in the migration already have the default `beeline-site.xml` file defined for the service. Other hosts do not. Configure these hosts as a gateway for that service.

Procedure

1. Find the notes you made before the upgrade about edge nodes and default, connected endpoints.
2. In Cloudera Manager, configure hosts other than HiveServer (HS2) hosts that you want to be Hive Gateway nodes as gateways for the default `beeline-site.xml` file for the gateway service.

Configure HiveServer HTTP mode

If you use Knox, you might need to change the HTTP mode configuration. If you installed Knox on Cloudera Base on premises and want to proxy HiveServer with Knox, you need to change the default HiveServer transport mode (`hive.server2.transport.mode`).

Procedure

1. Click Cloudera Manager Clusters HIVE_ON_TEZ Configuration
2. In Search, type `transport`.
3. In HiveServer2 Transport Mode, select `http`.

The screenshot shows the Cloudera Manager interface for configuring 'Hive on Tez'. The 'Configuration' tab is active, and a search for 'transport' has been performed. On the left, a 'Filters' sidebar shows the 'SCOPE' filter with 'Hive on Tez (Service-Wide)' selected, showing 1 instance. The main configuration area displays the 'HiveServer2 Transport Mode' section, where the 'hive.server2.transport.mode' property is set to 'http' (selected with a radio button). Other options like 'binary' and 'all' are unselected. The 'HiveServer2 Default Group' is set to 'Undo'. At the bottom, a message indicates '1 Edited Value' and a 'Save Changes(CTRL+S)' button is visible.

4. Save and restart Hive on Tez.

Handling syntax changes

You need to modify queries affected by changes to Hive syntax after upgrading to CDP. Hive has changed the syntax related to ``db.table`` references, such as `CREATE TABLE `mydb.mytable` ...`. Other syntax changes involve the `LOCATION` clause in `CREATE TABLE`. Hive in CDP supports the enhancement to `CREATE TABLE` that adds the `MANAGEDLOCATION` clause.

Handling table reference syntax

For ANSI SQL compliance, Hive 3.x rejects `db.table` in SQL queries as described by the Hive-16907 bug fix. A dot (.) is not allowed in table names. As a Data Engineer, you need to ensure that Hive tables do not contain these references before migrating the tables to Cloudera, that scripts are changed to comply with the SQL standard references, and that users are aware of the requirement.

About this task

To change queries that use such `db.table` references thereby preventing Hive from interpreting the entire db.table string incorrectly as the table name, you enclose the database name and the table name in backticks as follows:

A dot (.) is not allowed in table names.

Procedure

1. Find a table having the problematic table reference.
For example, 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));
```

LOCATION and MANAGEDLOCATION clauses

Before upgrading, your Hive version might have supported using the LOCATION clause in queries to create either managed or external tables or databases for managed and external tables. After upgrading, Hive stores managed and external tables in separate HDFS locations. CREATE TABLE limits the use of the LOCATION clause, and consequently requires a change to your queries. Hive in Cloudera also supports a new location-related clause.

External table limitation for creating table locations

Hive assigns a default location in the warehouse for external tables—/warehouse/tablespace/external/hive. In Cloudera, Hive does not allow the LOCATION clause in queries to create a managed table. Using this clause, you can specify a location only when creating external tables. For example:

```
CREATE EXTERNAL TABLE my_external_table (a string, b string)
ROW FORMAT SERDE 'com.mytables.MySerDe'
WITH SERDEPROPERTIES ( "input.regex" = "/*.csv" )
LOCATION '/warehouse/tablespace/external/hive/marketing';
```

Table MANAGEDLOCATION clause

In Cloudera, Hive has been enhanced to include a MANAGEDLOCATION clause to specify the location of managed tables as shown in the following syntax:

```
CREATE (DATABASE|SCHEMA) [IF NOT EXISTS] database_name
[COMMENT database_comment]
[LOCATION external_table_path]
[MANAGEDLOCATION managed_table_directory_path]
[WITH DBPROPERTIES (property_name=property_value, ...)];
```

Hive assigns a default location in the warehouse for managed tables—/warehouse/tablespace/managed/hive. In the MANAGEDLOCATION clause, you specify a top level directory for managed tables when creating a Hive database.



Important: Do not use the LOCATION clause to specify the location of managed tables. This clause is only used to specify the location of external tables. Use the MANAGEDLOCATION clause if you are creating managed tables. You must also ensure that you do not set LOCATION and MANAGEDLOCATION to the same HDFS path.

Use DESCRIBE DATABASE db_name; to view the root location of the database on the filesystem.

Related Information

[Create a default directory for managed tables](#)

Key semantic changes and workarounds

As SQL Developer, Analyst, or other Hive user, you need to know potential problems with queries due to semantic changes. Some of the operations that changed were not widely used, so you might not encounter any of the problems associated with the changes.

Over the years, Apache Hive committers enhanced versions of Hive supported in legacy releases of CDH and HDP, with users in mind. Changes were designed to maintain compatibility with Hive applications. Consequently, few syntax changes occurred over the years. A number of semantic changes, described in this section did occur, however. Workarounds are described for these semantic changes.

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 to Cloudera

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 to Cloudera

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.

Related Information

[Apache Hive web site summary of timestamp semantics](#)

Changing incompatible column types

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

Before Upgrade to Cloudera

In HDP 2.x and CDH 5.x and CDH 6 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 to Cloudera

In Cloudera, `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. Check ALTER TABLE statements and change those that would fail due to incompatible column types.

Understanding CREATE TABLE behavior

Hive table creation has changed significantly since Hive 3 to improve useability and functionality. If you are upgrading from CDH or HDP, you must understand the changes affecting legacy table creation behavior.

Hive has changed table creation in the following ways:

- Creates ACID-compliant table, which is the default in Cloudera
- 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. Using ACID-compliant, transactional tables causes no performance or operational overload. The way you access managed Hive tables from Spark and other clients changes. In Cloudera, access to external tables requires you to set up security access permissions.

You must understand the behavior of the CREATE TABLE statement in legacy platforms like CDH or HDP and how the behavior changes after you upgrade to Cloudera.

Before upgrading to CDP Private Cloud Base

In CDH 5, CDH 6, and HDP 2, by default CREATE TABLE creates a non-ACID managed table in plain text format.

In HDP 3 and CDP 7.1.0 through 7.1.7.x, by default CREATE TABLE creates either a full ACID transactional table in ORC format or insert-only ACID transactional tables for all other table formats.

After upgrading to CDP Private Cloud Base

- If you are upgrading from HDP 2, CDH 5, or CDH 6 to CDP 7.1.0 through CDP 7.1.8, by default CREATE TABLE creates a full ACID transactional table in ORC format or insert-only ACID transactional tables for all other table formats.
- If you are upgrading from HDP 3 or CDP 7.1.0 through 7.1.7.x to CDP 7.1.8, the existing behavior persists and CREATE TABLE creates either a full ACID transactional table in ORC format or insert-only ACID transactional tables for all other table formats.

Now that you understand the behavior of the CREATE TABLE statement, you can choose to modify the default table behavior by configuring certain properties. The order of preference for configuration is as follows:

Modify the default CREATE TABLE behavior

Override default behavior when creating the table

Irrespective of the database, session, or site-level settings, you can override the default table behavior by using the MANAGED or EXTERNAL keyword in the CREATE TABLE statement.

```
CREATE [MANAGED][EXTERNAL] TABLE foo (id INT);
```

Set the default table type at a database level

You can use the database property, `defaultTableType=EXTERNAL` or `ACID` to specify the default table type to be created using the `CREATE TABLE` statement. You can specify this property when creating the database or at a later point using the `ALTER DATABASE` statement. For example:

```
CREATE DATABASE test_db WITH DBPROPERTIES ( 'defaultTableType'='EXTERNAL' );
```

In this example, tables created under the `test_db` database using the `CREATE TABLE` statement creates external tables with the purge functionality enabled (`external.table.purge = 'true'`).

You can also choose to configure a database to allow only external tables to be created and prevent creation of `ACID` tables. While creating a database, you can set the database property, `EXTERNAL_TABLES_ONLY=true` to ensure that only external tables are created in the database. For example:

```
CREATE DATABASE test_db WITH DBPROPERTIES ( 'EXTERNAL_TABLES_ONLY'='true' );
```

Set the default table type at a session level

You can configure the `CREATE TABLE` behavior within an existing beeline session by setting `hive.create.as.external.legacy` to `true` or `false`. Setting the value to `true` results in configuring the `CREATE TABLE` statement to create external tables by default.

When the session ends, the default `CREATE TABLE` behavior also ends.

Set the default table type at a site level

You can configure the `CREATE TABLE` behavior at the site level by configuring the `hive.create.as.insert.only` and `hive.create.as.acid` properties in Cloudera Manager under Hive configuration.

When configured at the site level, the behavior persists from session to session. For more information, see [Configuring CREATE TABLE behavior](#).

If you are a Spark user, switching to legacy behavior is unnecessary. Calling `'create table'` from SparkSQL, for example, creates an external table after upgrading to Cloudera as it did before the upgrade. You can connect to Hive using the Hive Warehouse Connector (HWC) to read Hive `ACID` tables from Spark. To write `ACID` tables to Hive from Spark, you use the HWC and HWC API. Spark creates an external table with the purge property when you do not use the HWC API. For more information, see [Hive Warehouse Connector for accessing Spark data](#).

Related Information

[HDFS ACLS](#)

[Hive Warehouse Connector for accessing Apache Spark data](#)

[Spark Direct Reader for accessing Spark data](#)

[Apache Hive 3 Key Features](#)

[Apache Hive 3 Tables](#)

Configuring legacy CREATE TABLE behavior

After you upgrade to CDP Private Cloud Base and migrate old tables, the legacy `CREATE TABLE` behavior of Hive is no longer available by default and you might want to switch to the legacy behavior. Legacy behavior might solve compatibility problems with your scripts during data migration, for example, when running ETL.

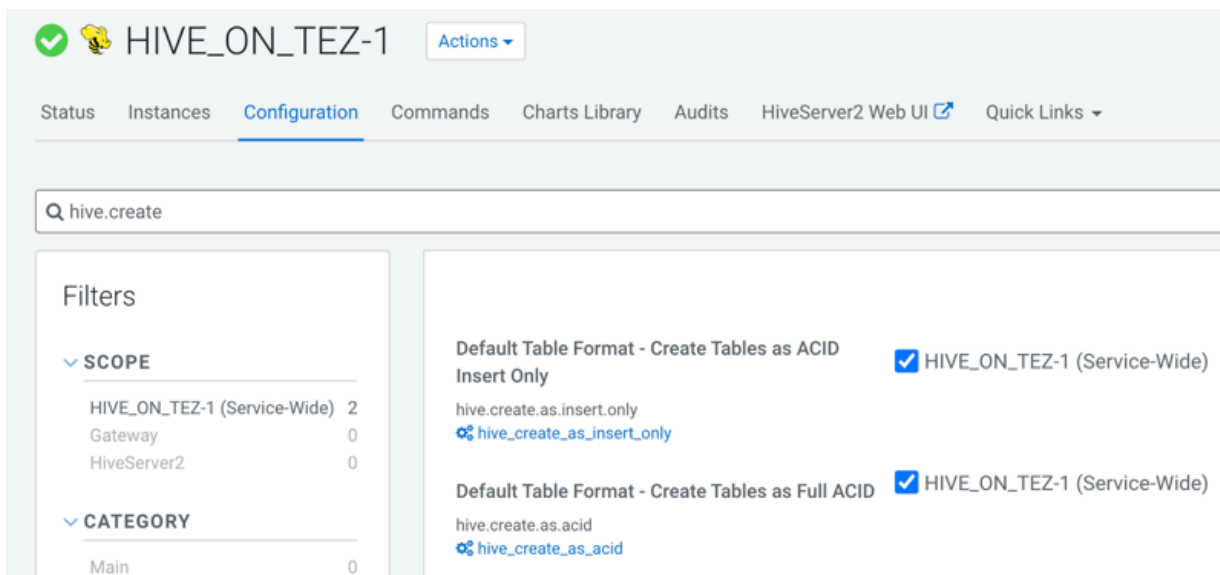
About this task

In Cloudera, running a `CREATE TABLE` statement by default creates a full `ACID` table for ORC file format and insert-only `ACID` table for other file formats. You can change the default behavior to use the legacy `CREATE TABLE` behavior. When you configure legacy behavior, `CREATE TABLE` creates external tables with the purge functionality enabled (`external.table.purge = 'true'`). Therefore, when the table is dropped, data is also deleted from the file system.

You can configure legacy `CREATE TABLE` behavior at the site level by configuring properties in Cloudera Manager. When configured at the site level, the behavior persists from session to session.

Procedure

1. In Cloudera Manager, click Clusters and select the Hive on Tez service.
2. From the Hive on Tez service, go to the Configuration tab and search for hive.create.



3. If the following properties are selected, clear the selection to enable legacy CREATE TABLE behavior.
 - Default Table Format - Create Tables as ACID Insert Only (hive.create.as.insert.only)
 - Default Table Format - Create Tables as Full ACID (hive.create.as.acid)

Results

Legacy behavior is enabled and the CREATE TABLE statement now creates external tables with the external.table.purge table property set to true.

Related Information

[Change DROP behavior](#)

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 to CDP Private Cloud Base

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

After Upgrade to CDP Private Cloud Base

OFFLINE and NO_DROP are not supported in the DROP CASCADE clause.

Action Required

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

Handling output of greatest and least functions

To calculate the greatest (or least) value in a column, you need to work around a problem that occurs when the column has a NULL value.

Before Upgrade to Cloudera

The greatest function returned the highest value of the list of values. The least function returned the lowest value of the list of values.

After Upgrade to Cloudera

Returns NULL when one or more arguments are NULL.

Action Required

Use NULL filters or the nvl function on the columns you use as arguments to the greatest or least functions.

```
SELECT greatest(nvl(col1,default value incase of NULL),nvl(col2,default value incase of NULL));
```

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 to Cloudera

In CDH and HDP, renaming a managed table moves its HDFS location.

After Upgrade to Cloudera

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

TRUNCATE TABLE on an external table

Hive 3 does not support TRUNCATE TABLE on external tables. Truncating an external table results in an error. You can truncate an external table if you change your applications to set a table property to purge data.

Before Upgrade to Cloudera

Some legacy versions of Hive supported TRUNCATE TABLE on external tables.

After Upgrade to CDP Private Cloud Base

By default, TRUNCATE TABLE is supported only on managed tables. Attempting to truncate an external table results in the following error:

```
Error: org.apache.spark.sql.AnalysisException: Operation not allowed: TRUNCATE TABLE on external tables
```

Action Required

Change applications. Do not attempt to run TRUNCATE TABLE on an external table.

Alternatively, change applications to alter a table property to set external.table.purge to true to allow truncation of an external table:

```
ALTER TABLE mytable SET TBLPROPERTIES ('external.table.purge'='true');
```

Migrating Spark Apps

Spark integration with Hive

You need to know a little about Hive Warehouse Connector (HWC) and how to find more information because to access Hive from Spark, you need to use HWC implicitly or explicitly.

You can use the Hive Warehouse Connector (HWC) to access Hive managed tables from Spark. HWC is specifically designed to access managed ACID v2 Hive tables, and supports writing to tables in Parquet, ORC, Avro, or Textfile formats. HWC is a Spark library/plugin that is launched with the Spark app.

You do not need HWC to read from or write to Hive external tables. Spark uses native Spark to access external tables.

Use the Spark Direct Reader and HWC for ETL jobs. For other jobs, consider using Apache Ranger and the HiveWarehouseConnector library to provide row and column, fine-grained access to the data.

HWC supports spark-submit and pyspark. The spark thrift server is not supported.

Related Information

[Hive Warehouse Connector for accessing Apache Spark data](#)

Identifying and fixing invalid Hive schema versions

As Administrator, after upgrading from Ambari-managed HDP to Cloudera Private Cloud Base, you need to identify Hive metastore operations that might fail due to Hive schema version incompatibility.

About this task

Incompatibility might exist if the upgrade process failed to make schema updates. You need to turn on the Hive Metastore Schema validation process for the metastore during the migration of your workloads to CDP. The Hive metastore captures any schema updates that occur during the upgrade, and displays issues in the Hive metastore logs. With this information, you can use the Apache Hive Schema tool to fix any problems.

Procedure

1. In Cloudera Manager, click **Clusters HIVE Configuration**.
2. Check the `hive.metastore.server.max.message.size`.



Max Message Size for Hive MetaStore

Hive Metastore Server Default Group [Undo](#)

hive.metastore.server.max.message.size

[hive_metastore_server_max_message_size](#)

100 MiB

3. Set `hive.metastore.server.max.message.size` to the recommended value: 10% of the value of your Java heap size for Hive Metastore Server in bytes, but no more than 21478364. Recommended value: 214748364
4. Click **Clusters HIVE Configuration**, and search for schema.
5. Check Strict Hive Metastore Schema Validation to set `hive.metastore.schema.validation` to true.
6. Check the Hive metastore logs and set a compatible metastore schema for the current Hive version using the Apache Hive Schema Tool.

Related Information

[Apache Hive Schema Tool](#)

Fixing statistics

Upgrading or migrating from Hive 1 or Hive 2 to Hive 3 might result in missing statistics. In Hive 3, these missing statistics, when detected by the cost-based optimizer (CBO), could cause datasets to be disregarded. As Data Engineer, you need to fix these statistics after upgrading.

Procedure

1. Run `DESCRIBE FORMATTED <table>`, and check the value of `numrows`.
If the value is 0, you must fix statistics.

2. Run ANALYZE on the tables and columns to fix the statistics.

```
ANALYZE TABLE credit_card_01.cc_acct COMPUTE STATISTICS[FOR COLUMNS];
```

Converting Hive CLI scripts to Beeline

If you have legacy scripts that run Hive queries from edge nodes using the Hive CLI, you must solve potential incompatibilities with variable substitution in these scripts. CDP 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 allowlist (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 run 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. Include these configuration parameters in the allowlist: In Cloudera Manager, go to Clusters `HIVE_ON_TEZ-1` Configuration, and search for `hive-site`.
7. In `HiveServer2 Advanced Configuration Snippet (Safety Valve)` for `hive-site.xml`, add the property key: `hive.security.authorization.sqlstd.confwhitelist.append`.
8. Provide the property value, or values, to allowlist, for example: `mylocal\..*|junk`.
This action appends `mylocal.test.var` and `mylocal.test.env.var` parameters to the allowlist.
9. Save configuration changes, and restart any components as required.
10. Run a command that references `HIVEVAR` to parse a SQL script, removes the incompatible `env` namespace, and runs 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  |
+-----+-----+
```

Hive unsupported interfaces and features

You need to understand the interfaces that are not supported.

Unsupported Interfaces and features

The following interfaces are not supported in Cloudera Base on premises:

- Druid
- Hcat CLI (however HCatalog is supported)
- Hive CLI (replaced by Beeline)
- Hive View UI feature in Ambari
- Apache Hive Standalone driver
- Renaming Hive databases
- Multiple insert overwrite queries that read data from a source table.
- LLAP
- MapReduce execution engine (replaced by Tez)
- Pig
- S3 for storing tables (available in Cloudera on cloud only)
- Spark execution engine (replaced by Tez)
- Spark thrift server

Spark and Hive tables interoperate using the Hive Warehouse Connector.

- SQL Standard Authorization
- Storage Based Authorization
- Tez View UI feature in Ambari
- WebHCat

You can use Hue in lieu of Hive View.

Storage Based Authorization

Storage Based Authorization (SBA) is no longer supported in Cloudera. Ranger integration with Hive metastore provides consistency in Ranger authorization enabled in HiveServer (HS2). SBA did not provide authorization support for metadata that does not have a file/directory associated with it. Ranger-based authorization has no such limitation.

Hive-Kudu integration

Cloudera does not support the integration of HiveServer (HS2) with Kudu tables. You cannot run queries against Kudu tables from HS2.

Partially unsupported interfaces

Apache Hadoop Distributed Copy (DistCP) is not supported for copying Hive ACID tables.

Unsupported Features

Cloudera does not support the following features that were available in HDP and CDH platforms:

- CREATE TABLE that specifies a managed table location

Do not use the LOCATION clause to create a managed table. Hive assigns a default location in the warehouse to managed tables. That default location is configured in Hive using the `hive.metastore.warehouse.dir` configuration property, but can be overridden for the database by setting the `CREATE DATABASE MANAGEDLOCATION` parameter.

- CREATE INDEX and related index commands were removed in Hive 3, and consequently are not supported in Cloudera.

In Cloudera, you use the Hive 3 default ORC columnar file formats to achieve the performance benefits of indexing. Materialized Views with automatic query rewriting also improves performance. Indexes migrated to Cloudera are preserved but render any Hive tables with an undroppable index. To drop the index, google the Known Issue for CDPD-23041.

- Hive metastore (HMS) high availability (HA) load balancing in CDH

You need to set up HMS HA as described in the documentation.

- Local or Embedded Hive metastore server

Cloudera does not support the use of a local or embedded Hive metastore setup.

Unsupported Connector Use

Cloudera does not support the Sqoop exports using the Hadoop jar command (the Java API) that Teradata documents. For more information, see [Migrating data using Sqoop](#).

Replicating Hive data from HDP 3 to CDP

You can replicate Hive ACID and external table data from an HDP 3.1.5.6000 cluster to a CDP Private Cloud Base 7.1.6 or higher cluster by applying patches, and then running the REPL DUMP command on the HDP cluster using a cron script. You run the REPL LOAD command on the CDP Private Cloud Base cluster using the Hive scheduler.

You follow step-by-step instructions to configure the HDP and CDP clusters. Configuration involves a number of policy-level properties and is mandatory. After replicating data, you follow detailed steps that describe how to verify the replication.



Note: If you want to use REPL commands to replicate Hive ACID tables between CDP Private Cloud Base clusters, ensure that your source cluster is on CDP Private Cloud Base 7.1.8 or a higher version.

Replicating Hive data

You need to meet a few prerequisites to perform this type of replication, apply patches, and configure the HDP and CDP clusters for replication. From example replication commands, you see how to replicate your data. Finally, you follow a step-by-step verification procedure.

Before you begin

- Ensure that the HDP cluster can write to the CDP cluster.
- Set up a one-way trust between the clusters if they belong to different Kerberos Key Distribution Centers (KDCs).
- To run the commands, log into beeline using kinit.

Procedure

1. Apply patches required on HDP.
2. Apply patches required on CDP.
3. Configure the clusters following step-by-step instructions.
4. Replicate Hive data as shown in example commands for replicating HDP 3 workloads.
5. Verify the Hive data replication.

Configuring the CDP cluster

You need to take advantage of Hive scheduled queries to load replicated workloads from HDP onto CDP using the REPL LOAD command. In the event of replication process problems, scheduled query metrics help you troubleshoot.

Before you begin

To perform Hive replication of external tables, add the hive user to the supergroup.

Procedure

1. Run a scheduled query on the CDP Private Cloud Base cluster to create a replication policy, using values for mandatory properties in the Mandatory CDP policy-level properties table in the next topic.

```
create scheduled query repl_['***replication policy name***',
['***FREQ***'] as REPL LOAD ['***SOURCE DB NAME***'] into ['***TARGET DB
NAME***'] with ['***Configuration parameters in key value pairs
separated by comma***'] executed as ['***user_name***'];
```

- Ensure that the replication policy name is in repl_['***policy name***'] format.
The scheduler is a generic scheduler in Hive and is used for various purposes including replication.
 - Make sure to filter the replication-related schedules.
2. Change the replication policy using the Hive statements in Supported Scheduled Query Operations.

Related Information

[Supported scheduled query operations](#)

Mandatory CDP policy-level properties

You need to configure mandatory Hive policies on CDP before you load workload replicated data. You learn what value to set for each property, whether or not you can specify property configuration options in the REPL LOAD command, and if the property is modifiable with an alter query in the REPL LOAD command.

The following table lists the properties you must set on the CDP Private Cloud Base cluster before loading your replicated data onto CDP. The Modifiable column indicates whether or not you can modify the property with an alter query in the REPL LOAD command.

Table 4: Mandatory Policy-level Properties

Property	Description	Required Value	Modifiable
hive.repl.rootdir	Staging location path. Enter the same path for REPL DUMP and REPL LOAD.	[***HDFS path***] Note: Use the same property value in REPL DUMP and REPL LOAD.	No
hive.repl.include.external.tables	Includes external tables for replication.	true	No
hive.repl.dump.metadata.only.for.external.table	Includes only external table metadata for replication.	false	yes
hive.repl.replica.external.table.base.dir	Fully qualified base directory on the target warehouse to store external tables. The directory path is prefixed to the source external table path on the target cluster. Enter the same path for REPL DUMP and REPL LOAD commands.	[***HDFS path***] Note: Use the same property value in REPL DUMP and REPL LOAD commands.	yes
hive.repl.ha.datapath.replace.remote.nameservice	Set to true when the following are true: - HDFS is HA-enabled - Both the source and target clusters are configured with the same nameservice name.	See Description before you configure this option.	Yes
hive.repl.ha.datapath.replace.remote.nameservice.name	Provides a reference to the nameservice on HDP when HDFS is HA-enabled and both HDP and CDP clusters are configured with the same nameservice name.	[**remote nameservice name**] Note: Use the same parameter value in REPL DUMP and REPL LOAD commands.	Yes

The property values of the nameservice.name and remote.nameservice must be different. For example, if the clusters use the nameservice name ns, then use a different property value. For example, 'hive.repl.ha.datapath.replace.remote.nameservice.name' = 'nsRemote'.



Important: Do not configure hive.repl.ha.datapath.replace.remote.nameservice and hive.repl.ha.datapath.replace.remote.nameservice.name configuration parameters if the clusters are non-HA or have different nameservice names.

Optional CDP policy-level properties

You must include the following list of optional policy-level configuration properties in REPL LOAD command on the CDP Private Cloud Base cluster. Put the property after the WITH clause of the command.

The Modifiable column indicates whether or not you can modify the property with an alter query in the REPL LOAD command.

Table 5:

Property	Description	Default Value	Modifiable
distcp.options.pugbx	Enter the options based on whether you want to preserve owner or user permissions, group permissions, and HDFS ACLs in source and target clusters during replication. Note: You must have superuser privileges to preserve the user and group permissions, and HDFS ACLs.	N/A	Yes
hive.repl.retry.initial.delay	First retry delay in seconds.	60 seconds	yes
hive.repl.retry.backoff.coefficient	Exponential Delay between retries. The value of (Previous Delay) * (Backoff Coefficient) determines the next retry interval.	1.2 seconds	Yes
hive.repl.retry.jitter	A Random jitter to be applied to avoid all retries happening at the same time.	30 seconds	Yes
hive.repl.retry.max.delay.between.retries	Maximum allowed retry delay in seconds after including the exponential backoff algorithm. If this limit is reached, retry will continue with this duration.	60 minutes	Yes
hive.repl.retry.total.duration	Total allowed retry duration in seconds inclusive of all retries. After this is duration is exceeded, the policy instance is marked as failed and will need manual intervention to restart.	24 hours	Yes

Supported scheduled query operations

When you configure the CDP cluster, you can change the replication policy using the Hive scheduled queries that CDP supports.

CDP supports the following scheduled queries that you use to change the scheduled query that replicates the HDP workload to CDP.

- Modify the replication policy schedule:

```
alter scheduled query repl_policyname cron '2 2 * * *';
```

- Run the replication policy:

```
alter scheduled query repl_policyname execute;
```

- Delete the replication policy:

```
drop scheduled query repl_policyname;
```

- Pause running the replication policy job:

```
alter scheduled query repl_policyname disabled;
```


- Resume running the replication policy job:

```
alter scheduled query repl_policyname enabled;
```

- List scheduled replication policies:

```
select * from sys.scheduled_queries;
```

- List all scheduled runs for all policies:

```
select * from sys.scheduled_executions;
```

- Track the error for the last job run for a replication policy:

```
select s.error_message from sys.scheduled_executions s join sys.schedule
d_queries t where s.scheduled_query_id = 7
t.scheduled_query_id and t.schedule_name = '<policy-name>' order by s.sch
eduled_execution_id limit 1;
```

- Get the metrics from the Hive scheduler for a replication policy:

```
select * from sys.replication_metrics where policy_name=repl<policy name>
order by scheduled_execution_id desc limit 1;
```

When you get metrics from the Hive scheduler, in case of irrecoverable errors, the status shows **FAILED_ADMIN** and the error log path appears in the metrics. Manually check the error and delete the error file to resume replication after you fix the error. For more information, see [Troubleshooting](#).

The following sample snippet shows the Hive scheduler metrics on the target CDP cluster:

```
+-----+-----+-----+-----+
+-----+-----+-----+-----+
+-----+-----+-----+-----+
+
| replication_metrics.scheduled_execution_id | replication_metrics.policy
_name | replication_metrics.dump_execution_id | replication_metrics.metadat
a | replication_metrics.progress
|
+-----+-----+-----+-----+
+-----+-----+-----+-----+
+-----+-----+-----+-----+
-+
| 2380 | 0 | repl_db1
| {"dbName":"dbl_r","replica
tionType":"BOOTSTRAP","stagingDir":"hdfs://ns1/
user/hive/replDir/dl/ZGIx/fddc6e66-4ec4-431b-92ab-826c94bba5/hive","lastRepl
Id":12082}
|
| {"status":"SUCCESS","stages":[{"name":"REPL_LOAD",
"status":"SUCCESS","startTime":1617669672695,"endTime":1617669675068,"met
rics":[{"name":"FUNCTIONS","currentCount":0,"totalCount":0},{ "name":"TABLES"
,"currentCount":2,"totalCount":2}], "errorLogPath":null}} |
```

Related Information

[Configuring the CDP cluster](#)

Configuring the HDP cluster

You need to configure the HDP cluster before you dump workload data that you want to replicate on CDP.

Before you begin

Prepare a cron script to set policies for chained REPL DUMP commands and to control execution, for example to run at a certain time.

Procedure

1. In new and existing databases, include the repl.source.for property in the source database dbproperties file.

Set the repl.source.for property value using the following format:

```
'repl.source.for' = [****policy1 name***, ****policy2 name***, ****policy3 name***]
```

For example, to create a new source database for policies named 1, 2, and 3, configure the source database properties file as follows:

```
'repl.source.for' = '1, 2, 3'
```

For example, to configure an existing source database named testdb, run the following command:

```
ALTER DATABASE testdb SET
DBPROPERTIES('repl.source.for'=[****policy1 name, policy2 name,
policy3 name***]);
```

2. On the HDP cluster, configure the mandatory HDP cluster configuration properties listed in the next topic.
3. Run the REPL DUMP command along the mandatory policy-level configuration parameters using a cron script.

Use the following command syntax:

```
[**cron syntax for regular intervals**] beeline -u jdbc:hive2://[**source database**] hive
-e"repl dump [**source database**] with [**mandatory policy-level configuration
parameters separated by comma**]
```

See the Cron Expression Generator & Explainer website.

Related Information

[Cron Expression Generator & Explainer](#)

Mandatory HDP cluster configuration properties

You must set the mandatory cluster-level properties on the HDP cluster before replicating data using the REPL command. You need to set the values required for a successful replication.

Table 6:

Property	Description	Required Value
hive.repl.cm.enabled	Enables ChangeManager so that the deleted files are saved in the cmrootdir directory.	true
hive.repl.cmroot	Root directory (cmrootdir directory) for ChangeManager which is used for deleted files.	[**HDFS path**]
hive.repl.cm.retain	Time in days to retain the deleted files in cmrootdir directory.	10 days
hive.metastore.event.db.listener.timetolive	Time in days after which the events are removed from the database listener queue.	10 days

Mandatory HDP policy-level properties

You must set a number of policy-level properties in REPL DUMP command on the HDP cluster. Put the property after the WITH clause of the command.

Table 7:

Property	Description	Required Value
hive.repl.dump.version	REPL DUMP format version. - Use 2 for HDP-to-CDP Private Cloud Base replication. - Use 1 for HDP to HDP replication.	2
hive.repl.rootdir	Staging location path. Enter the same path for REPL DUMP and REPL LOAD commands.	[**HDFS path**] Note: Use the same parameter value in REPL DUMP and REPL LOAD commands.
hive.repl.include.external.tables	Include external tables for replication.	true
hive.repl.dump.metadata.only.for .external.table	Includes only external table metadata for replication.	false
hive.repl.replica.external.table.b ase.dir	Fully qualified base directory on the target warehouse to store external tables. The directory path is prefixed to the source external table path on the target cluster.	[**HDFS path**] Note: Use the same parameter value in REPL DUMP and REPL LOAD commands.
hive.repl.ha.datapath.replace.re mote.nameservice	Set to true when the following are true: - HDFS is HA-enabled - Both the source and target clusters are configured with the same nameservice name.	See Description before you configure this option.
hive.repl.ha.datapath.replace.re mote.nameservice.name	Provides a reference to the nameservice on the remote (target) cluster when HDFS is HA-enabled and both source and target clusters are configured with the same nameservice name.	[**remote nameservice name**] Note: Use the same parameter value in REPL,DUMP and REPL LOAD commands.

hive.repl.ha.datapath.replace.re mote.nameservice.name value requirements

Ensure the value of hive.repl.ha.datapath.replace.re mote.nameservice.name is different from the nameservice name on the local (source) cluster. For example, if the clusters use the nameservice name ns, then use a different property value. For example, 'hive.repl.ha.datapath.replace.remote.nameserv ice.name' = 'nsRemote'.



Important: Do not configure hive.repl.ha.datapath.replace.remote.nameservice and hive.repl.ha.datapath.re place.remote.nameservice.name properties if the clusters are non-HA or have different nameservice names.

Optional HDP policy-level properties

You must include optional policy-level configuration properties in REPL DUMP command on the HDP cluster. You put the property after the WITH clause of the command.

Table 8:

Property	Description	Default Value
distcp.options.pugbx	Enter the options based on whether you want to preserve owner or user permissions, group permissions, and HDFS ACLs in source and target clusters during replication. Note: You must have superuser privileges to preserve the user and group permissions, and HDFS ACLs.	N/A
hive.repl.retry.initial.delay	First retry delay in seconds.	60 seconds
hive.repl.retry.backoff.coefficient	Exponential Delay between retries. The value of (Previous Delay) * (Backoff Coefficient) determines the next retry interval.	1.2 seconds
hive.repl.retry.jitter	A Random jitter to be applied to avoid all retries happening at the same time.	30 seconds

Configuring wire-encrypted clusters

If CDP clusters are wire-encrypted (TLS-enabled), you need to know how to configure the clusters for running the Apache Hadoop DistCp (Distributed Copy) command.

About this task

To run DistCp on a wire-encrypted (TLS-enabled) multi-cluster environment, perform the following steps to export the Ranger KMS certificate from the RangerKMS host of the HDP cluster to the Hadoop client truststore of CDP Private Cloud Base cluster.

Procedure

1. Run the following CLI command to export the certificate from Ranger KMS keystore file on KMS hosts of both the clusters.

```
cd [***kms_key_store_location***];keytool -export-alias kms_cert_[***host_name***]-keystore [***kms_keystore_file_path***] -rfc -filekms_cert_[***host_name***] -storepass[***kms_keystore_password***]
```

The `ranger.https.attr.keystore.file` parameter in the KMS configuration file contains the location of the KMS keystore.

2. Copy all the certificates generated for KMS in the HDP cluster to the client key location on all the hosts of the CDP Private Cloud Base cluster. Similarly, copy all the certificates generated for KMS in the CDP Private Cloud Base cluster to the client key location on all the hosts of the HDP cluster.
3. Run the following CLI command to import all the KMS certificates in the HDP cluster to the Hadoop client truststore on all the hosts of CDP Private Cloud Base cluster.

```
cd [***client_hadoop_key_location***];keytool import-noprompt -aliaskms_cert_[***host_name***] -file kms_cert_[***host_name***]-keystore[***truststore_file_path***] -storepass [***truststore_password***]
```

4. Similarly, import all the KMS certificates in the CDP Private Cloud Base cluster to the Hadoop client truststore on all the hosts of the HDP cluster.
5. Restart the HDFS service, YARN, MapReduce, and RangerKMS on both the clusters.

Related Information

[SSL configuration for Distcp across the cluster in wire encrypted Multicluster Environment](#)

Example commands for replicating HDP 3 workloads

To perform Hive replication from an HDP 3.1.5.6000 cluster to a CDP Private Cloud Base 7.1.6 or later cluster, you need to know how to run the REPL DUMP on the HDP cluster and REPL LOAD on the CDP Private Cloud Base cluster. Examples of valid commands helps you create counterparts to replicate your workloads to CDP. You learn how to replicate data between HA clusters.

Example Hive commands for replicating managed tables

To replicate managed tables, use the REPL commands as shown in these examples.

1. On HDP, dump a workload of managed tables.

```
repl dump src with (
'hive.repl.dump.version'= '2',
'hive.repl.rootdir'= 'hdfs://<host>:<port>/user/hive/replDir/d1'
);
```

2. On CDP, load the workload of managed tables.

```
repl load src into tgt with (
'hive.repl.rootdir'= 'hdfs://<host>:<port>/user/hive/replDir/d1'
);
```

Example Hive commands for replicating external tables

Prerequisite: To perform Hive replication of external tables, add the hive user to the supergroup.

1. On HDP, dump a workload of external tables.

```
repl dump src with (
'hive.repl.dump.version'= '2',
'hive.repl.rootdir'= 'hdfs://<host>:<port>/user/hive/replDir/d1',
'hive.repl.include.external.tables'= 'true',
'hive.repl.dump.metadata.only.for.external.table'= 'false',
'hive.repl.replica.external.table.base.dir'=
'hdfs://<replica-host>:<port>/user/hive/externalDir/d1'
);
```

2. On CDP, load the external tables.

```
repl load src into tgt with (
'hive.repl.dump.version'= '2',
'hive.repl.rootdir'= 'hdfs://<host>:<port>/user/hive/replDir/d1',
'hive.repl.include.external.tables'= 'true',
'hive.repl.dump.metadata.only.for.external.table'= 'false',
'hive.repl.replica.external.table.base.dir'=
'hdfs://<replica-host>:<port>/user/hive/externalDir/d1'
);
```

Example Hive commands for replicating data between HA clusters

To perform Hive replication between HA clusters (HDP and CDP Private Cloud Base clusters), you must provide HDFS-related HA configuration properties for HDP and CDP clusters in the REPL DUMP and REPL LOAD commands. The following examples show Hive replication from an HDP cluster to a CDP Private Cloud Base cluster that are HA-enabled.

1. On HDP, dump a workload.

```
repl dump src with (
```

```

'hive.repl.dump.version'= '2',
'hive.repl.rootdir'= 'hdfs://ns1/user/hive/replDir/d1',
'hive.repl.dump.metadata.only.for.external.table'= 'false',
'hive.repl.replica.external.table.base.dir'=
  'hdfs://ns1/user/hive/externalDir/d1',
'hive.repl.include.external.tables'= 'true',
'dfs.nameservices'= 'mycluster,ns1',
'mapreduce.job.hdfs-servers.token-renewal.exclude'= 'ns1',
'dfs.ha.automatic-failover.enabled'= 'true',
'dfs.ha.namenodes.mycluster'= 'nn1,nn2',
'dfs.namenode.rpc-address.mycluster.nn1'=
  'ctr-1617214704777-622-01-007.h.site:8020',
'dfs.namenode.rpc-address.mycluster.nn2'=
  'ctr-1617214704777-622-01-004.h.site:8020',
'dfs.namenode.http-address.mycluster.nn1'=
  'ctr-1617214704777-622-007.h.site:20070',
'dfs.namenode.http-address.mycluster.nn2'=
  'ctr-1617214704777-622-01-004.h.site:20070',
'dfs.namenode.https-address.mycluster.nn1'=
  'ctr-1617214704777-622-01-007.h.site:20470',
'dfs.namenode.https-address.mycluster.nn2'=
  'ctr-1617214704777-622-01-004.h.site:20470',
'dfs.client.failover.proxy.provider.mycluster'=
  'org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxy
Provider',
'dfs.ha.automatic-failover.enabled.ns1'= 'true',
'dfs.ha.namenodes.ns1'= 'namenode1546333583,namenode1546336357',
'dfs.namenode.rpc-address.ns1.namenode1546333583'= 'sar-uk-1.sar-uk.root
.h.site:8020',
'dfs.namenode.rpc-address.ns1.namenode1546336357'= 'sar-uk-2.sar-uk.root.h
.site:8020',
'dfs.namenode.http-address.ns1.namenode1546333583'= 'sar-uk-1.sar-uk.roo
t.h.site:20101',
'dfs.namenode.http-address.ns1.namenode1546336357'= 'sar-uk-2.sar-uk.roo
t.h.site:20101',
'dfs.namenode.https-address.ns1.namenode1546336357'= 'sar-uk-1.sar-uk.ro
ot.h.site:20102',
'dfs.namenode.https-address.ns1.namenode1546333583'= 'sar-uk-1.sar-uk.root.
h.site:20102',
'dfs.client.failover.proxy.provider.ns1'=
  'org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProx
yProvider'
);

```

2. On CDP, load the workload.

```

repl load src into tgt with (
'hive.repl.rootdir'= 'hdfs://ns1/user/hive/replDir/d1',
'hive.repl.dump.metadata.only.for.external.table'= 'false',
'hive.repl.replica.external.table.base.dir'=
  'hdfs://ns1/user/hive/externalDir/d1',
'hive.repl.include.external.tables'= 'true',
'dfs.nameservices'= 'mycluster,ns1',
'mapreduce.job.hdfs-servers.token-renewal.exclude'= 'mycluster',
'dfs.ha.automatic-failover.enabled'= 'true',
'dfs.ha.namenodes.mycluster'= 'nn1,nn2',
'dfs.namenode.rpc-address.mycluster.nn1'=
  'ctr-1617214704777-622-01-007.h.site:8020',
'dfs.namenode.rpc-address.mycluster.nn2'=
  'ctr-1617214704777-622-01-004.h.site:8020',
'dfs.namenode.http-address.mycluster.nn1'=
  'ctr-1617214704777-622-01-007.h.site:20070',
'dfs.namenode.http-address.mycluster.nn2'=
  'ctr-1617214704777-622-01-004.h.site:20070',

```

```
'dfs.namenode.https-address.mycluster.nn1'=
  'ctr-1617214704777-622-01-007.h.site:20470',
'dfs.namenode.https-address.mycluster.nn2'=
  'ctr-1617214704777-622-01-004.h.site:20470',
'dfs.client.failover.proxy.provider.mycluster'=
  'org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProx
yProvider',
'dfs.ha.automatic-failover.enabled.ns1'= 'true',
'dfs.ha.namenodes.ns1'= 'namenode1546333583,namenode1546336357',
'dfs.namenode.rpc-address.ns1.namenode1546333583'='sar-uk-1.sar-uk.root.h.
site:8020',
'dfs.namenode.rpc-address.ns1.namenode1546336357'='sar-uk-2.sar-uk.root.
h.site:8020',
'dfs.namenode.http-address.ns1.namenode1546333583'=
  'sar-uk-1.sar-uk.root.h.site:20101',
'dfs.namenode.http-address.ns1.namenode1546336357'='sar-uk-2.sar-uk.roo
t.h.site:20101',
'dfs.namenode.https-address.ns1.namenode1546336357'='sar-uk-2.sar-uk.ro
ot.h.site:20102',
'dfs.namenode.https-address.ns1.namenode1546333583'='sar-uk-1.sar-uk.root.
h.site:20102',
'dfs.client.failover.proxy.provider.ns1'=
  'org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProx
yProvider'
);
```

Troubleshooting Hive replication using REPL

You need to know how to recover from the FAILED_ADMIN state that stops the replication process.

Problem: A non-recoverable error appears for a replication job and the status says FAILED_ADMIN. How do you recover a schedule from the FAILED_ADMIN state?

Solution: Perform the following steps to recover a replication schedule from this state:

1. Navigate to the error log path.
2. Search for the file `_non_recoverable`.
3. Open the file, and look for information about an error that caused the replication failed.
4. Fix the error.
5. Delete the `_non_recoverable` file.

The `_non_recoverable` file from the last replication command execution must be deleted; otherwise your replication attempt will malfunction.

Problem: Notification events are missing in the metastore.

Solution: If notification events are not present in the metastore during replication, the replication might be in a FAILED_ADMIN status. When this occurs, notifications are deleted in the metastore. In this case, the workaround is to start a fresh bootstrap phase of replication, as follows:

1. Drop the target database using beeline.
2. Remove the dump directory on HDFS for the required policy. The path of `_non_recoverable` error file path has the dump directory path.

The replication continues where it stopped.

Repl Command Known Issues

You need to know the about REPL command issues that you might encounter when replicating Hive data from HDP to CDP Private Cloud Base in CDP 7.1.6 and earlier.

- **HIVE-24896**

Incremental replication fails when a managed table is dropped and an external table is created with the same name.

- **HIVE-24933**

Incremental replication fails when an external table is dropped and a managed table is created with the same name.

- **HIVE-24818**

Replication fails when there are partitioned views.

- **HIVE-24676**

Show locks call failing for PostgreSQL database if the txnId filter is passed.

- **HIVE-24698**

The ACLs are not synchronized for external tables automatically.

- **HIVE-24878**

Function replication is not initiated if the `hive.repl.run.data.copy.tasks.on.target` policy-level configuration parameter is set to false.

Workaround: By default, the `hive.repl.run.data.copy.tasks.on.target` policy-level configuration parameter is set to true. Before you initiate function replication, ensure that this parameter is set to true.

- **CDPD-23188**

Replication of Hive data fails when the external tables in the HDP cluster are replicated to a managed warehouse location in the CDP Private Cloud Base cluster.

This issue might appear when one of the following condition is true:

- The external tables in HDP are in the managed warehouse location but the `hive.repl.replica.external.table.base.dir` parameter is set to “/”.
- The `hive.repl.replica.external.table.base.dir` parameter is configured with the managed warehouse location in the CDP Private Cloud Base cluster.

The Managed warehouse location is defined by the `hive.metastore.warehouse.dir` configuration parameter.

Workaround: Perform the following steps to resolve the issue:

- Configure the `hive.repl.replica.external.table.base.dir` parameter to a value other than “/”. This ensures that the external tables are not replicated to the managed warehouse location on target.
- Configure the `hive.repl.replica.external.table.base.dir` parameter with a path other than the managed warehouse location in CDP Private Cloud Base cluster.

Patches Required on HDP

Before replicating HDP 3.1.5 databases to CDP 7.1.6, you apply JIRAs on HDP-3.1.5.6000 on top of HDP-3.1.5.0. A list of these JIRAs includes links to more information about each patch.

- **CDPD-27374: HIVE-20823: Make Compactor run in a transaction.**
- [HOTFIX-3686](#) **BUG-124830 HIVE-21036** extend OpenTxnRequest with transaction type
- [HOTFIX-3686](#) **BUG-124830 HIVE-22367:** Transaction type not retrieved from OpenTxnRequest
- [HOTFIX-3686](#) **BUG-124830 HIVE-21114:** Create read-only transactions
- [HOTFIX-3686](#) **BUG-124830 HIVE-22327:** Repl: Ignore read-only transactions in notification log
- [HOTFIX-3685](#) **BUG-124830 HIVE-23340** TxnHandler cleanup
- [HOTFIX-3686](#) **BUG-124830 HIVE-23560:** Optimize bootstrap dump to abort only write Transactions
- [HOTFIX-3686](#) **BUG-124830 HIVE-24095:** Load partitions in parallel for external tables in the bootstrap phase
- [HOTFIX-3686](#) **BUG-124830 HIVE-24328:** Run distcp in parallel for all file entries in repl load.

- [HOTFIX-3686](#) BUG-124830 HIVE-22290: ObjectStore.cleanWriteNotificationEvents and ObjectStore.cleanupEvents OutOfMemory on large number of pending events
- [HOTFIX-3686](#) BUG-124830 HIVE-24197: Check for write transactions for the db under replication at a frequent interval
- [HOTFIX-3686](#) BUG-124830 HIVE-24109: Load partitions in batches for managed tables in the bootstrap phase
- [HOTFIX-3686](#) BUG-124830 HIVE-24363: Current order of transactional event listeners is prone to deadlock in backend DB connections
- [HOTFIX-3686](#) BUG-124830 HIVE-23851: MSCK REPAIR Command With Partition Filtering Fails While Dropping Partitions (#1271)
- ENGESC-363: Backport [HIVE-22736](#): Support multiple encryption zones in hive replication
- ENGESC-363 : [HIVE-22890](#) : Fix Repl load failure if table name contains _function
- [HIVE-22844](#): Validate cm configs, add retries in fs apis for cm.
- [CDPD-20160](#): Add new repl dump format support on HDP
- [HIVE-24597](#). Replication with timestamp type partition failing in HA case with same NS.
- [HIVE-24432](#): Delete Notification Events in Batches.
- [HIVE-24676](#) : Show locks call failing for postgres RDBMS if the txnId filter is passed
- [BUG-125039](#) : HL_TXNID column is lower case in HDP-3.1.5
- [CDPD-20160](#): Add new repl dump format support on HDP - incremental
- [CDPD-21491](#): HIVE-24675: Handle external table replication for HA with same NS and lazy copy
- [HIVE-24127](#): Dump events from default catalog only
- [BUG-125093](#): Test Failures in DBNotification Listener and ACID
- [BUG-124718](#) HIVE-24127: Dump events from default catalog only
- [CDPD-23113](#): HDP_TO_CDP: Repl_dump is failing fo Staging_dir=src_cluster
- [CDPD-23204](#): incremental function replication failing for data copy on source
- [CDPD-23206](#): HIVE-24856: Skip functions created without 'using' clause during incremental replication
- [HIVE-24836](#). Add replication policy name and schedule id as a job name for all the distep jobs
- [HIVE-24895](#). Add a DataCopyEnd stage in ReplStateLogTask for external table replication. Partial Backport only contains per task logging
- [HIVE-24909](#): Skip the repl events from getting logged in notification log
- [HIVE-25272](#): Read transactions are getting logged in the notification log

Patches required on CDP

Before replicating HDP 3.1.5 workloads to CDP 7.1.6, you apply several patches to the CDP cluster.

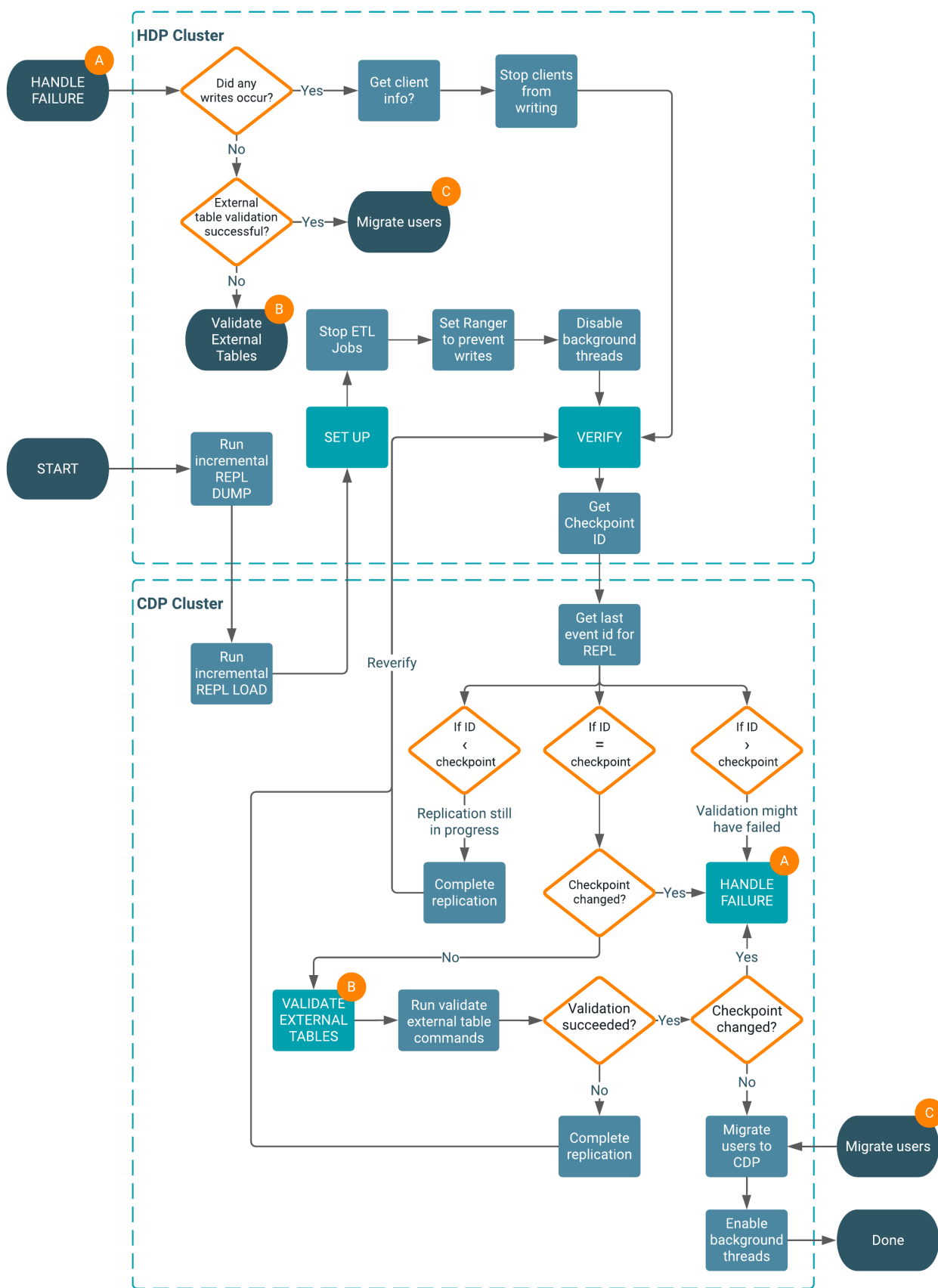
- HIVE-25002 Modify condition for target of replication in statsUpdaterThread and PartitionManagementTask.
- HIVE-24881 Abort old replication transactions.
- HIVE-25218 External table verification tool.

Verifying the Hive data replication from HDP 3.1.5 to 7.1.6

After replicating Hive databases from HDP 3.1.5 (with patches) to CDP 7.1.6 (with patches) you need to verify that the replication succeeded. After verification that databases on both clusters are in sync, you can stop running workloads on the HDP cluster and restart workloads on the CDP cluster.

About this task

To verify the replication, follow steps to stop ETL jobs and set up Ranger policies on HDP. Disable HDP background threads, and get an event marker for validation later. Finally, you validate the replication, or handle possible replication failures. The following diagram shows the replication and verification process.



Setting up the HDP cluster

You set up the HDP cluster after replicating one or more databases before you can verify replication. Set up requires stopping jobs.

About this task

Of course, before you can verify a replication, you must have completed one. The first cycle in replicating data is called the bootstrap, and the following cycles are called incremental replications. For each database, a completed replication consists of one bootstrap and at least one incremental replication cycle.

Before you begin

- Run at least one incremental replication for a databases before attempting verification.
- In the CDP cluster, find the dump directory path using the following query:

```
select * from sys.replication_metrics
where policy_name='<policy name>'
order by scheduled_execution_id desc limit 1;
```

- Find and copy the external table paths listed in the CDP dump directory path in `_file_list_external` file. You will use these paths to set up Ranger policies in Ambari.

Procedure

1. On the HDP source cluster, stop all ETL jobs.

- 2. In Ambari Ranger Admin Service Manager Hive policies , add a Deny policy (no writes) for all users including 'hive' on all databases: Database *, Table *, Hive column *

You need only one policy to deny any writes to managed tables or any access to any external tables

Ranger

Access Manager

Audit

Security Zone

Settings

Service Manager

cl1_hive Policies

Edit Policy

Edit Policy

Policy Details :

Policy Type

Access

Policy ID

30

Policy Name *

migration_deny

enabled

Policy Label

Policy Label

database

*

*

include

table

*

*

include

Hive Column *

*

*

include

Deny Conditions :

Select Group

public

hiveuser

Select User

hiveuser

user1

user2

hive

Permissions

update

Create

Drop

Alter

Write

- 3. In Ambari Ranger Admin Service Manager HDFS policies , add a Ranger Deny policy for all external table paths.

4. In Resource Path, paste the external table paths you copied from in the CDP dump directory path in the `_file_list_external` file.

You can add single or multiple policies for all the external table paths in all the databases.

For example:

5. Disable the StatsUpdaterThread background thread by configuring the `hive.metastore.stats.auto.analyze` property to none.
6. Disable the PartitionManagementTask background thread by configuring the `metastore.partition.management.database.pattern` property to `^*`.

Verifying replication

To verify the replication, you get a checkpoint event id on the HDP cluster and the last event id. Based on your comparison of the ids, you complete the verification, repeat the verification, or re-replicate the data.

Before you begin

You have run the replication policies, and think you have replicated all databases.

Procedure

1. On the HDP cluster backend RDBMS for Hive metastore (HMS), run a query to get a checkpoint event id.

```
select NEXT_EVENT_ID - 1 as last_event_id from NOTIFICATION_SEQUENCE;
```

Output looks something like this:

```
MariaDB [hivedb]> select NEXT_EVENT_ID - 1 as last_event_id from NOTIFICATION_SEQUENCE;
+-----+
| last_event_id |
+-----+
|          582 |
+-----+
1 row in set (0.00 sec)
```

You use the HDP

last event id, 582 in this example, as a checkpoint id later.

- On the CDP cluster, run the following query to get the last event id for each database replication.

```
select name as database_name, param_value as last_event_id
      from sys.dbs join sys.database_params on dbs
      .db_id = database_params.db_id
      where param_key='repl.last.id'
```

For example, as root you invoke Hive with the Hive 3-supported option `-e`, and enclose the query exactly as shown above in quotation marks.

Output looks something like this:

```
[root@quasar-wpjgk-3 ~]# hive -e "select name as database_name, param_value as last_event_id f
rom sys.dbs join sys.database_params on dbs.db_id = database_params.db_id where param_key='repl
.last.id'"
```

```
+-----+-----+
| database_name | last_event_id |
+-----+-----+
| db1           | 582           |
| db2           | 582           |
| a_repl        | 582           |
+-----+-----+
3 rows selected (10.323 seconds)
```

- Check the last event ID that appears in the output.
 - If the last event ID equals the HDP checkpoint id, the replication process succeeded for the database. Go to the next step in this procedure.
 - If the last event id is greater than the checkpoint id, verification might be incomplete. Proceed to handling a failed verification.
 - If the last event id is less than the checkpoint id, the replication is still in progress. Continue the replication for that database until the checkpoint id is reached, and reverify.
- On the HDP cluster backend RDBMS for Hive metastore (HMS), check that the checkpoint id has not changed since the last time you recorded it. For example

```
select NEXT_EVENT_ID - 1 as last_event_id from NOTIFICATION_SEQUENCE;
```

Output looks something like this:

```
MariaDB [hivedb]> select NEXT_EVENT_ID - 1 as last_event_id from NOTIFICATION_SEQUENCE;
+-----+
| last_event_id |
+-----+
| 582           |
+-----+
1 row in set (0.00 sec)
```

- Based on your comparison of the `last_event_id` and the checkpoint ID, proceed in one of the following ways:
 - If the HDP `last_event_id` does not match the checkpoint id you recorded, go to the next topic, "Handling a failed verification";
 - If the HDP `last_event_id` does match the checkpoint id, proceed to the next step.
- Repeat these steps to verify the replication of each database in the workload, and then validate external table replication.

Handling a failed verification

Although you stopped ETL jobs and set Ranger policies to prevent writes to databases, a database write still might occur. This causes a verification failure. You need to find out if such a write caused the failure.

About this task

Before you begin

- You completed the replication of Hive data from HDP to CDP.
- The last event id in the output is at least the checkpoint id for all databases, but does not match the checkpoint id.
- You noted the HDP notification event id, which you use later as a checkpoint id.

Procedure

1. On the backend RDBMS of the HDP cluster, run a query to determine if any writes occurred during replication.

```
select * from NOTIFICATION_LOG where event_id > <HDP notification event id
> and event_type = 'COMMIT_TXN';
```

This query gets all commit transaction (TXN) events after the HDP checkpoint event-id.

If there are no commit events, the output looks something like this:

```
MariaDB [hivedb]> select * from NOTIFICATION_LOG where event_id > 582 and event_type = 'COMMIT_TXN';
Empty set (0.00 sec)
```

If there are no commit events, assume all valid writes are replicated. Skip the rest of the steps. The migration is done.

If there are commit events, continue to the next step.

2. Run a query to get information about the writes.

```
select event_type,db_name, count(*) from NOTIFICATION_LOG
where event_id > <HDP notification event id >
group by db_name,event_type order by db_name;
```

The output might look something like this:

```
MariaDB [hivedb]> select event_type,db_name, count(*) from NOTIFICATION_LOG where event_id > 582
2 group by db_name,event_type order by db_name;
+-----+-----+-----+
| event_type | db_name | count(*) |
+-----+-----+-----+
| ABORT_TXN  | NULL    | 4         |
| OPEN_TXN   | NULL    | 4         |
| ALLOC_WRITE_ID_EVENT | db1     | 2         |
| ALLOC_WRITE_ID_EVENT | db2     | 2         |
+-----+-----+-----+
4 rows in set (0.00 sec)
```

3. Identify clients who wrote to the database, and stop further attempts.
You can use Ranger audits, if enabled, to identify such clients as described in Managing Auditing.
4. Repeat the procedure "Verifying replication".

Validating external table replication

You need to validate external table replication before migration to CDP. You run the external table validation commands on the CDP cluster.

About this task

You run the external table validation commands using the command syntax and options below:

Command Syntax

```
hive --replMigration -dumpFilePath <path to external table info file> \
[-dirLevelCheck] [-fileLevelCheck] [-verifyOpenFiles] [-verifyChecksum] [-f
ilters] \
[-conf] [-queueSize] [-numThreads]
```

Options

- `-dumpFilePath`: The fully qualified path to the external table info file.
- `-dirLevelCheck`: Validate at directory level.
- `-fileLevelCheck`: Validate at file level.
- `-verifyOpenFiles`: Validate there are no open files on the source path. Requires superuser privileges.
- `-verifyChecksum`: Whether the checksum needs to be validated for each file. Cannot be used with `-dirLevelCheck`. Will fail if the source and target are in different encryption zones or use different checksum algorithms.
- `-filters`: Comma separated list of filters, cannot be used along with `-dirLevelCheck`.
- `-conf`: Semi-Colon separated list of additional configurations in `key1=value1;key2=value2` format.
- `-queueSize`: Queue size for the thread pool executor for table level validation. Default: 200
- `-numThreads`: Number of threads for thread pool executor for table level validation. Default: 10
- `-checksumQueueSize`: Queue size for the thread pool executor for checksum computation. Default: 200
- `-checksumNumThreads`: Number of threads for thread pool executor for checksum computation. Default: 5

Running the external table validation commands can take a significant amount of time, and writes to the HDP database can occur. You need to check the checkpoint ID again after validating external table replication to determine if any writes did indeed happen.

To validate external table replication, perform the following steps:

Procedure

1. In CDP, run the external table validation commands as described above.
2. If validation is successful, proceed to the next step; otherwise, complete the replication.
External table validation fails if external data has not been fully replicated; another replication and verification cycle is required to sync the data.
3. On the HDP cluster backend RDBMS for Hive metastore (HMS), check that the checkpoint id has not changed since the last time you recorded it. For example:

```
select NEXT_EVENT_ID - 1 as last_event_id from NOTIFICATION_SEQUENCE;
```

Output looks something like this:

```
MariaDB [hivedb]> select NEXT_EVENT_ID - 1 as last_event_id from NOTIFICATION_SEQUENCE;
+-----+
| last_event_id |
+-----+
|          582 |
+-----+
1 row in set (0.00 sec)
```

4. Based on your comparison of the `last_event_id` and the checkpoint ID, proceed in one of the following ways:
 - If the HDP `last_event_id` does not match the checkpoint id you recorded, handle a failed verification.
 - If the HDP `last_event_id` does match the checkpoint id, the validation is successful. Migrate users to CDP and enable background threads. You are done.

Enabling background threads after migration

Before replication, you disabled background threads from running on databases. After migration, you need to enable background threads to run.

Procedure

1. In the CDP cluster, run a query to enable background threads to run on a replicated database.

```
alter database <dbName> set dbproperties('repl.target.for'='')
```

2. Repeat the last step on each replicated database to enable background threads to run.

You can now stop using the HDP cluster and start working on the CDP cluster.

Migration paths from HDP 3 to CDP for LLAP users

If you are running your Hive HDP 3.x workloads using LLAP (low-latency analytical processing), you need to decide on the best migration path to CDP without compromising on the performance offered by LLAP. Migration recommendations depend on a number of factors, such as your workload type and whether you have Hive or Spark users.

Migration paths for Hive users

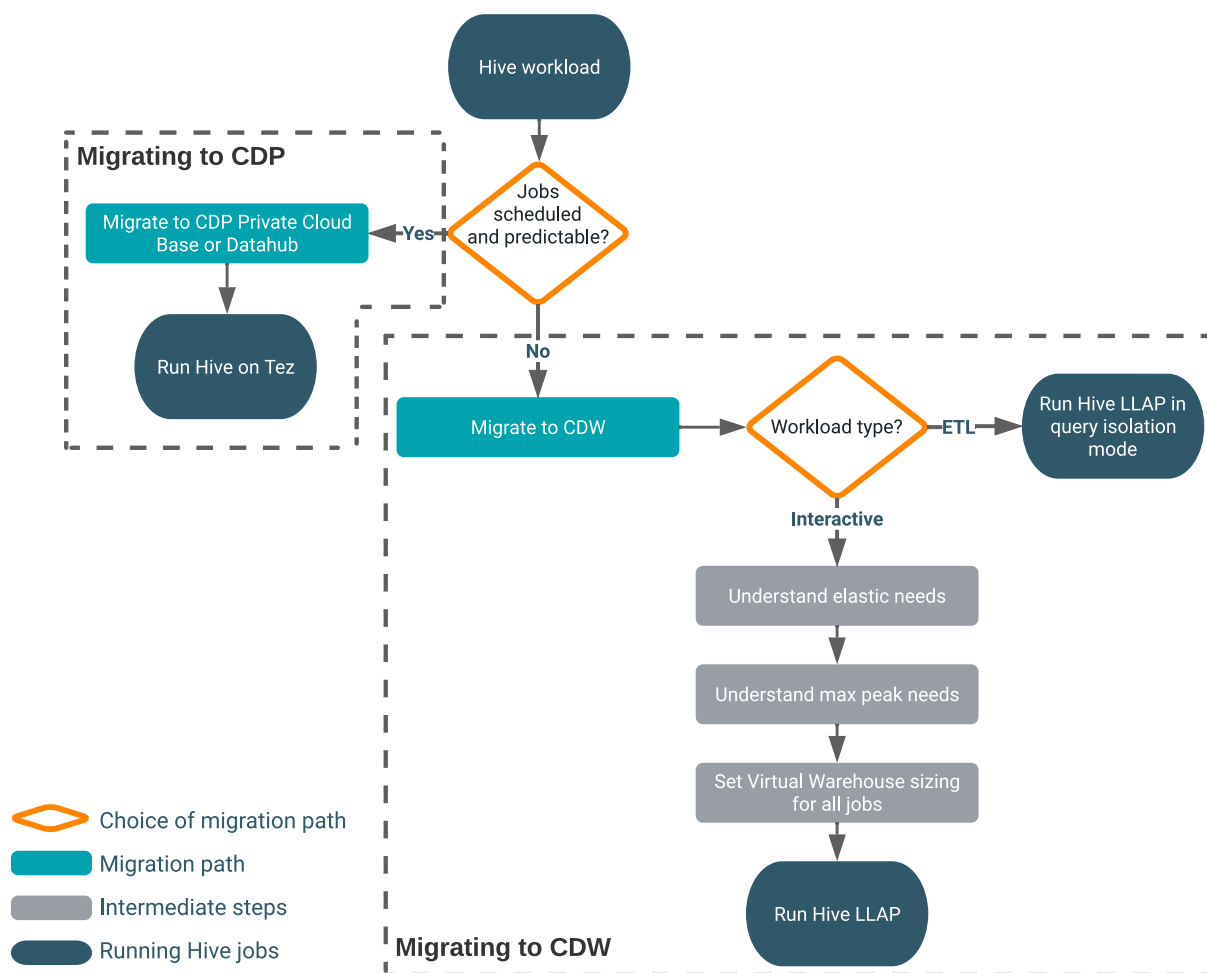
If you are on HDP and executing Hive workloads in Hive LLAP mode and want to upgrade to CDP, you can follow the migration path that matches your use case.

Using LLAP is recommended for running complex ETL jobs that are unpredictable. LLAP is offered as part of Cloudera Data Warehouse (CDW) and is not available in CDP Public Cloud (Data Hub) and Cloudera Private Cloud Base. Using Hive on Tez (no LLAP) is recommended for running jobs that are scheduled and predictable. Hive on Tez is supported on CDP Public Cloud and Cloudera Private Cloud Base.

The following migration paths are recommended based on how predictable your jobs are and your workload type:

- Migrate to CDP Public Cloud (Data Hub) or Cloudera Private Cloud Base — If your jobs are scheduled and predictable.
- Migrate to Cloudera Data Warehouse (CDW) — If your jobs are unpredictable and if they could increase demand on compute resources.

The following diagram shows these recommended paths:



Migration to Cloudera Private Cloud Base or CDP Public Cloud

If your Hive jobs are scheduled, or otherwise considered predictable, consider migrating to Cloudera Private Cloud Base or CDP Public Cloud and running your Hive workloads on Hive on Tez.

You learn why you should consider the predictability of your jobs when choosing a migration path from HDP 3 to CDP. If you were using LLAP on HDP, a path to CDP can offer equivalent performance and might make sense cost-wise.

LLAP is not supported in Cloudera Private Cloud Base and CDP Public Cloud, but Hive-on-Tez performs well for jobs that are not subjected to unexpected big spikes in demand that would require elasticity, such as rapid service scaling. Predictable jobs do not require the immediate deployment of extra resources to respond to unexpected traffic to your cluster. You do not need to shut down resources when traffic drops suddenly.

Related Information

[Apache Tez execution engine for Apache Hive](#)

Migration to Cloudera Data Warehouse

If your Hive jobs are unpredictable, consider migrating to Cloudera Data Warehouse (CDW) and running your Hive workloads on LLAP mode. LLAP offers ETL performance and scalability for complex data warehousing jobs.

Such a move can meet or exceed your customer satisfaction requirements, or cut the expense of running your jobs, or both. You set up automatic scaling up of compute resources when needed, or shutting down when not needed.

LLAP along with the query isolation feature is best suited for data-intensive queries, such as ETL queries, and require auto-scaling based on the total scan size of the query.

If your Hive jobs are unpredictable and if you are running complex interactive queries, migrate to CDW. You perform the following configuration when taking this path:

- Configure CDW according to your plan for scaling and concurrency
- Tune the Hive Virtual Warehouse to handle peak workloads

When you create a CDW Virtual Warehouse, you configure the size and concurrency of queries to set up LLAP. The configuration is simple compared to HDP configurations of LLAP. In Hive Virtual Warehouses, each size setting indicates the number of concurrent queries that can be run. For example, an X-Small Hive on LLAP Virtual Warehouse can run 2 TB of data in its cache.

Related Information

[Apache Tez execution engine for Apache Hive](#)

[Query isolation for data-intensive queries](#)

[Auto scaling of Virtual Warehouses in CDW](#)

[Virtual Warehouse sizing requirements](#)

[Tuning Virtual Warehouses](#)

Apache Tez processing of Hive jobs

After migrating to Cloudera Private Cloud Base CDP Public Cloud, or Cloudera Data Warehouse (CDW), you must understand how the Apache Tez execution engine is used to run your Hive workloads.

Apache Tez provides the framework to run a job that creates a graph with vertices and tasks. The entire execution plan is created under this framework. Apache Tez provides the following execution modes:

- Container mode — Every time you run a Hive query, Tez requests a container from YARN.
- LLAP mode — Every time you run a Hive query, Tez asks the LLAP daemon for a free thread, and starts running a fragment.

SQL syntax in Hive is the same irrespective of execution mode used in Hive. In Cloudera Private Cloud Base and CDP Public Cloud, Tez always runs in container mode and is commonly called Hive on Tez. In CDW, Tez always runs in LLAP mode. You can use the query isolation feature if you are running complex ETL workloads.

Migration paths for Spark users

If you are on HDP and executing Spark workloads in Hive LLAP mode, and you want to upgrade to CDP, you can follow the migration path that matches your security needs. It is recommended that you upgrade to Cloudera Private Cloud Base and choose either Hive Warehouse Connector (HWC) or native Spark readers to query Hive from Spark.

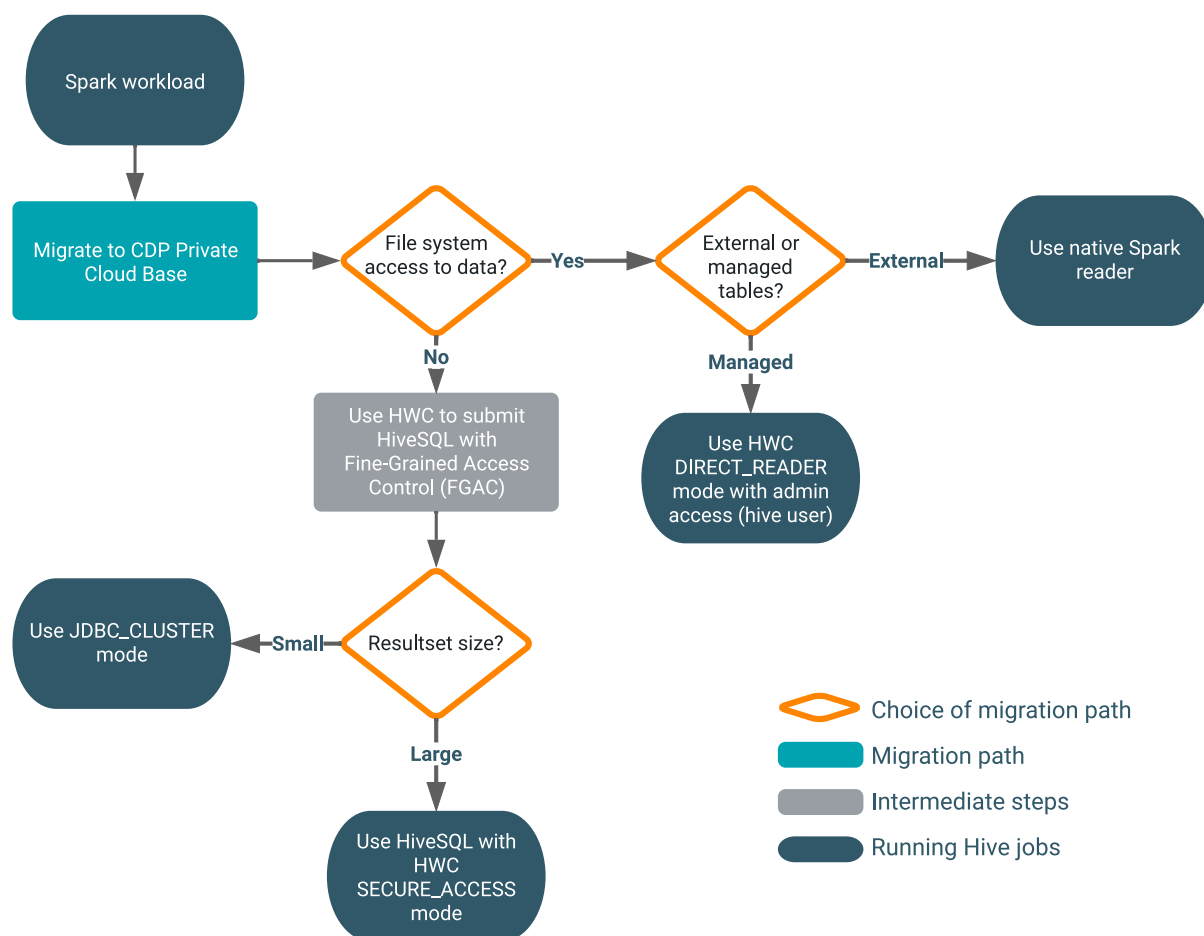
As a replacement for the HWC LLAP execution mode in HDP, you can use the HWC Secure Access Mode in Cloudera Private Cloud Base that offers fine-grained access control (FGAC) column masking and row filtering to secure managed (ACID), or even external, Hive table data that you read from Spark.

The following migration paths are recommended based on certain factors:

Migrate to Cloudera Private Cloud Base

- Use HWC JDBC Cluster mode — If the user does not have access to data in the file system and if the database query returns are less than 1 GB of data.
- Use HWC Secure access mode — If the user does not have access to data in the file system and if the database query returns are more than 1 GB of data.
- Use HWC Direct reader mode — If the user has access to data in the file system and if you are querying Hive managed tables.
- Use native Spark reader — If the user has access to data in the file system and if you are querying Hive external tables.

The following diagram shows these recommended paths:



Migration to Cloudera Private Cloud Base

If you are a Spark user migrating from HDP to CDP and accessing Hive workloads through the Hive Warehouse Connector (HWC), consider migrating to Cloudera Private Cloud Base and based on your use cases, use the various HWC read modes to access Hive managed tables from Spark.

Use HWC JDBC Cluster mode

If the user does not have access to the file systems (restricted access), you can use HWC to submit HiveSQL from Spark with benefits of fine-grained access control (FGAC), row filtering and column masking, to securely access Hive tables from Spark.

However, if the size of your database query returns are less than 1 GB of data, it is recommended that you use HWC JDBC Cluster mode in which Spark executors connect to Hive through JDBC, and execute the query. Larger workloads are not recommended for JDBC reads in production due to slow performance.

Use HWC Secure access mode

If the user does not have access to the file systems (restricted access) and if the size of database query returns are greater than 1 GB of data, it is recommended to use HWC Secure access mode that offers fine-grained access control (FGAC), row filtering and column masking to access Hive table data from Spark.

Secure access mode enables you to set up an HDFS staging location to temporarily store Hive files that users need to read from Spark and secure the data using Ranger FGAC.

Use HWC Direct reader mode

If the user has access to the file systems (ETL jobs do not require authorization and run as super user) and if you are accessing Hive managed tables, you can use the HWC Direct reader mode to allow Spark to read directly from the managed table location.



Important: This workload must be run with 'hive' user permissions.

If you are querying Hive external tables, use Spark native readers to read the external tables from Spark.

Related Information

[Row-level filtering and column masking in Hive](#)

[Introduction to HWC JDBC read mode](#)

[Introduction to HWC Secure access mode](#)

[Introduction to HWC Direct reader mode](#)

HWC changes from HDP to CDP

You need to understand the Hive Warehouse Connector (HWC) changes from HDP to CDP. Extensive HWC documentation can prepare you to update your HWC code to run on CDP. In CDP, methods and the configuration of the HWC connections differ from HDP.

Deprecated methods

The following methods have been deprecated in Cloudera Runtime 7.1.7:

- `hive.execute()`
- `hive.executeQuery()`

The HWC interface is simplified in CDP resulting in the convergence of the `execute/executeQuery` methods to the `sql` method. The `execute/executeQuery` methods are deprecated and will be removed from CDP in a future release. Historical calls to `execute/executeQuery` used the JDBC connection and were limited to 1000 records. The 1000 record limitation does not apply to the `sql` method, although using JDBC cluster mode is recommended only for production for workloads having a data size of 1GB or less. Larger workloads are not recommended for JDBC reads in production due to slow performance.

Although the old methods are still supported in CDP for backward compatibility, refactoring your code to use the `sql` method for all configurations (JDBC client, Direct Reader V1 or V2, and Secure Access modes) is highly recommended.

Recommended method refactoring

The following table shows the recommended method refactoring:

API	From HDP	To CDP	HDP Example	CDP Example
HWC sql API	<code>execute</code> and <code>executeQuery</code> methods	<code>sql</code> method	<code>hive.execute("select * from default.hwctest").show(1, false)</code>	<code>hive.sql("select * from default.hwctest").show(1, false)</code>
Spark sql API	<code>sql</code> and <code>spark.read.table</code> methods	No change	<code>sql("select * from managedTable").show</code> <code>scala> spark.read.table("managedTable").show</code>	<code>sql("select * from managedTable").show</code> <code>scala> spark.read.table("managedTable").show</code>
DataFrames API	<code>spark.read.format</code> method	No change	<code>val df = spark.read.format("HiveAcid").options(Map("tabl</code>	<code>val df = spark.read.format("HiveAcid").options(Map("tabl</code>

API	From HDP	To CDP	HDP Example	CDP Example
			e" -> "default.acidtbl").load()	e" -> "default.acidtbl").load()

Deprecated and changed configurations

HWC read configuration is simplified in CDP. You use a common configuration for Spark Direct Reader, JDBC Cluster, or Secure Access mode.

The following HWC configurations that you might have used in HDP 3.1.5 cannot be used in CDP:

- `--conf spark.hadoop.hive.llap.daemon.service.hosts`
- `--conf spark.hadoop.hive.zookeeper.quorum`

Recommended configuration refactoring

Refactor configuration code to remove unsupported configurations. Use the following common configuration property: `spark.datasource.hive.warehouse.read.mode`.

You can transparently read data from Spark with HWC in different modes using just `spark.sql("<query>")`.

Secured cluster configurations

In secured cluster configurations, you must set configurations as described in the HWC documentation for CDP:

- `--conf "spark.security.credentials.hiveserver2.enabled=true"`
- `--conf "spark.sql.hive.hiveserver2.jdbc.url.principal=hive/_HOST@ROOT.HWX.SITE"`

The jdbc url must not contain the jdbc url principal and must be passed as shown here.

Deprecated features

The following features have been deprecated and will be removed from CDP in a future release:

- Catalog browsing
- JDBC client mode configuration

Related Information

[Introduction to Hive Warehouse Connector](#)

Migrating Hive workloads from Cloudera Base on premises to Cloudera Data Warehouse on premises

To migrate the Hive workloads to Cloudera Data Warehouse Data Service, you must have upgraded from your legacy platform to Cloudera Base on premises. Learn how to set up your Virtual Warehouse instance and migrate your workloads to Hive LLAP (Low-Latency Analytical Processing) in Cloudera Data Warehouse.

The Cloudera Data Warehouse service provides data warehouses that can be configured and isolated. You scale resources up and down to meet your business demands, and save costs by suspending and resuming resources automatically.

Hive LLAP along with the query isolation feature is best suited for data-intensive queries, such as ETL queries, that require auto-scaling based on the total scan size of the query.

Migrating your Hive workloads to Cloudera Data Warehouse helps you leverage the auto-scaling, workload optimization, isolation, data caching, and many other powerful capabilities that Cloudera Data Warehouse offers.

This document aims to help you understand the process of migrating Hive workloads from Cloudera Base on premises to Cloudera Data Warehouse and assumes that you have already upgraded to Cloudera Base on premises.

Review the following migration scenarios if you want to migrate to Cloudera Data Warehouse from legacy platforms or Cloudera:

- **Migrating from HDP to Cloudera Data Warehouse**

If you are migrating from HDP and running workloads using LLAP or Hive on Tez, you must first upgrade to Cloudera on premises either through an [In-Place upgrade](#) or [Sidecar migration](#). You can then follow the Migrating from Cloudera Base on premises to Cloudera Data Warehouse document to migrate your Hive workloads to Cloudera Data Warehouse.

To understand how LLAP in HDP is different from LLAP in Cloudera Data Warehouse, see [Migrate from HDP \(LLAP\) to Cloudera Data Warehouse \(LLAP\)](#).

- **Migrating from CDH to Cloudera Data Warehouse**

If you are migrating from CDH and running workloads using Hive on Map Reduce, you must first upgrade to Cloudera on premises either through an [In-Place upgrade](#) or [Sidecar migration](#). You can then follow the Migrating from Cloudera on premises to Cloudera Data Warehouse document to migrate your Hive workloads to Cloudera Data Warehouse.

- **Migrating from Cloudera Base on premises to Cloudera Data Warehouse**

If you are migrating from Cloudera on premises and running workloads in the Tez container mode (Hive on Tez), learn how you can migrate to Cloudera Data Warehouse on premises and run Hive workloads on LLAP mode. For more information, see [Migrate from Cloudera on premises \(Hive on Tez\) to Cloudera Data Warehouse \(LLAP\)](#).

After you have upgraded to Cloudera on premises, you can install Cloudera Data Services on premises and migrate your workloads to Cloudera Data Warehouse. Use the guidance provided in this document to plan your Virtual Warehouse instances based on the workloads that you are running. For more information, see [Planning a Virtual Warehouse instance](#).

You must also understand how Hive queries are processed in Cloudera Data Warehouse using the LLAP execution mode and be aware of the differences between LLAP in Cloudera Data Warehouse and Hive on Tez in Cloudera. For more information, see [Apache Tez processing of Hive jobs](#).

Related Information

[Installing on premises Data Services](#)

Planning a Cloudera Data Warehouse Virtual Warehouse instance

A Hive Virtual Warehouse is an instance of compute resources with various options that allow you to control the size, elasticity, and availability of the data warehouse to meet your varying workload demands. Therefore, it is essential to understand the type of workloads or number of concurrent queries that your Virtual Warehouse must serve during peak periods before deciding the size of your Virtual Warehouse.

If you are upgrading from legacy platforms or Cloudera to Cloudera Data Warehouse on premises and migrating workloads that used to run on YARN, you may want to consider building your Virtual Warehouse instance as a copy of the compute resources that you have defined in the YARN queues. If these YARN queues are sized based on workload types, such as BI or ETL, you may use this document as a reference to plan your Virtual Warehouse instance.

In Cloudera Data Warehouse, a Virtual Warehouse is categorized based on sizes that represent the size of an Executor Group that handles query requests. Queries can only run within the boundaries of an Executor Group.

Mixing workloads with different characteristics in the same Executor Group makes it difficult to predict Service Level Agreements (SLAs). Therefore, it is important that you understand your workloads and plan your Virtual Warehouses based on the workload types.

- **Business Intelligence (BI) workloads**

Business Intelligence (BI) workloads are usually smaller with more targeted datasets. Response times need to be optimized because operations depend on near real-time analysis.

- **Research BI workloads**

Research BI workloads dive deeper into data exploration than BI workloads. Datasets used in research may not always be optimized for performance. In Research BI workloads, the balance can tilt more towards the cost side of the equation than that of BI workloads.

- **Discovery ETL workloads**

Discovery ETL workloads are unpredictable and take “Research BI” into longer-running exploratory pipelines that might yield desired results.

- **Production ETL workloads**

Production ETL workloads are predictable and usually come with SLAs. They build out the data models used by BI and Research that run the business. They can also be very resource intensive and traverse through a lot of data.

You must create a Virtual Warehouse instance with these workload characteristics in mind. Data Services, where Virtual Warehouse instances are created, provide isolation for other Virtual Warehouses or Data Service instances running workloads. However, there is no isolation within an Executor Group of a Virtual Warehouse instance.

The following table lists the Virtual Warehouse sizes that you can choose based on the workload types that are described above:

	XSMALL (2 executors)	SMALL (10 executors)	MEDIUM (20 executors)	LARGE (40 executors)	Custom
BI	X	X			
Research BI		X	X		
Discovery ETL		X	X	X	
Production ETL			X1	X1	X1

1 Consider the Query Isolation mode for Virtual Warehouse instances that run complex ETL-type queries requiring intensive data scanning.

Related Information

[Hive auto-scaling on Cloudera Data Warehouse on premises](#)

[Hive LLAP auto-scale threshold settings](#)

Apache Tez processing of Hive jobs

If you were running Hive on HDP or Cloudera, you have been running Hive queries using the Apache Tez execution engine. Hive in Cloudera Data Warehouse on premises also uses Tez to run queries and is a HiveServer2 endpoint as it is in HDP or Cloudera. Learn how Tez processes Hive jobs in Cloudera and Cloudera Data Warehouse and understand the tasks that you need to perform after migrating your workloads to Cloudera Data Warehouse.

Hive is fundamentally the same technology in HDP, Cloudera Base on premises, and Cloudera Data Warehouse on premises. Hive syntax and semantics are basically the same after upgrading from HDP to Cloudera on premises or to Cloudera Data Warehouse on premises.

Apache Tez provides the framework to run a job that creates a graph with vertices and tasks. SQL semantics for deciding the query physical plan, which identifies how to execute the query in a distributed fashion, is based on Apache Tez. The entire execution plan is created under this framework. Apache Tez provides the following execution modes:

- Container mode — Every time you run a Hive query, Tez requests a container from YARN.
- LLAP mode — Every time you run a Hive query, Tez asks the LLAP daemon for a free thread, and starts running a fragment.

In Cloudera Data Warehouse, the Hive execution mode is LLAP. In Cloudera Data Hub on Cloudera on cloud and Cloudera Base on premises, the Hive execution mode is container, and LLAP mode is not supported. When Apache Tez runs Hive in container mode, it has traditionally been called Hive on Tez.

Considerations

There are certain differences between Hive on Tez and LLAP that you need to be aware of before migrating to Cloudera Data Warehouse on premises.

- The HiveServer2 endpoints authenticate using LDAP instead of Kerberos.
- Your old Hive JDBC drivers need to be replaced with the latest drivers.
- If you have Hive User-Defined Functions (UDFs) in Cloudera Base on premises then the UDF JARs have to be added to the Cloudera Data Warehouse Hive classpath and registered.

Post-migration tasks

After migrating to Cloudera Data Warehouse on premises, perform the following tasks:

1. Download the latest Hive JDBC drivers from the [Hive JDBC driver download page](#) and follow the driver installation instructions on the download page.
2. Update the JDBC client connection URL to point to the Virtual Warehouse instance of HiveServer2.
3. If your previous connection in Cloudera Base on premises used Kerberos for authentication, you must modify the connection URL accordingly.
4. Ensure that the UDF JARs are added to the CDW_HIVE_AUX_JARS_PATH environment variable.

Related Information

[Connecting Hive to BI tools using a JDBC driver in Cloudera Data Warehouse](#)

[Hive authentication](#)

[Uploading additional JARs to Cloudera Data Warehouse](#)

Migrate Hive workloads from HDP (LLAP) to Cloudera Data Warehouse (LLAP)

If you are on the HDP platform and running your Hive workloads using LLAP (low-latency analytical processing), learn how you can migrate to Cloudera Data Warehouse on premises without compromising on the performance offered by LLAP.

You perform the following high-level tasks to migrate from HDP (LLAP) to Cloudera Data Warehouse (LLAP):

1. Upgrade your clusters to Cloudera Base on premises.
2. Install Cloudera Data Services on premises.
3. Migrate your Hive workloads to Cloudera Data Warehouse.
4. Perform the post-migration tasks described in [Apache Tez processing of Hive jobs](#).

LLAP in HDP

LLAP on HDP runs on YARN with a persistent LLAP daemon that provides execution and caching of data. You can adjust many aspects of the LLAP deployment, such as:

- Size of the LLAP daemons (Memory / Executors)
- Number of daemons created to scale up and handle large workloads
- Number of Apache Tez ApplicationMasters (coordinators) to establish query concurrency
- Ratio of memory used for processing and cache

The YARN configurations in HDP are complex and are not usually optimal, leading to poor experiences.

In HDP, you could only have one LLAP instance running and the instance was sized to handle workloads at peak intervals. The LLAP instances in HDP could not autoscale and consumed a finite amount of YARN resources regardless of whether workloads were running or not.

When Hive LLAP is used for large complex ETL queries and without a robust workload management in place, large workloads can block BI-type workloads thereby leading to poor user experience for BI users.

Some advanced LLAP on YARN implementations may have used ‘Hive Workload Management’ in LLAP to help manage query isolation and deal with query outliers. ‘Hive Workload Management’ in LLAP is not supported in Cloudera Data Warehouse, however, query isolation in Cloudera Data Warehouse ensures that individual warehouses are completely isolated and ensures that instances have sufficient compute resources for their workloads.

LLAP in Cloudera Data Warehouse

In Cloudera Data Warehouse on premises, Hive LLAP runs in Docker containers on Kubernetes instead of YARN. The Virtual Warehouse instances in Cloudera Data Warehouse are preconfigured to handle the LLAP configurations described above and are optimized for your workloads thereby enabling a predictable and stable deployment.

Cloudera Data Warehouse offers the following benefits:

- Provides isolation by having more than one LLAP instance, which was not easily obtained in LLAP on HDP
- Enables Virtual Warehouse instances to AutoScale (scale up and down) to address varying workload demands
- Automatically suspends a Virtual Warehouse instance if workloads cease and Executors are left idle for a period of time

While setting up your Virtual Warehouse, consider setting up multiple instances to leverage the full use of LLAP that you have enjoyed on HDP. The Virtual Warehouse instances should be configured based on the characteristics of the workloads.

Query concurrency in Cloudera Data Warehouse is controlled by the number of Coordinators in an Executor Group. HiveServer locates an available query coordinator in the Virtual Warehouse to handle the query. The query coordinator generates the final query plan that distributes query tasks across available Executors for execution. Each query coordinator can send query tasks to all query Executors in the Executor Group. There is a 1:1 ratio of Coordinators to Executor Groups. When the query load increases, auto-scaling increases concurrency by adding additional query Executor Groups to the Virtual Warehouse instance.

The following table lists the size of the Executor Groups and Coordinators based on the Virtual Warehouse size:

	Executors	Coordinators (Query Concurrency)
XSMALL	2	2
SMALL	10	10
MEDIUM	20	20
LARGE	40	40

HDP and Cloudera Data Warehouse LLAP terminology map

When migrating HDP LLAP configurations to Cloudera Data Warehouse, you need to familiarize yourself with the differences in terminology and default configurations that are available in Cloudera Data Warehouse.

Term	LLAP on HDP	LLAP on Cloudera Data Warehouse
Number of LLAP nodes	num_llap_count	Number of query executors
Number of internal executors in a single LLAP daemon instance.	hive.llap.daemon.num.executors	Not available during setup. By default, 12 Executors are allowed per Executor Group.

Term	LLAP on HDP	LLAP on Cloudera Data Warehouse
Also described as the number of daemons; not the internal number of task slots.		
Query Concurrency	hive.server2.tez.sessions.per.default.queue	Known as Coordinators and are configured 1:1 with Executor Groups
Total Daemon Memory Size	hive.llap.daemon.yarn.container.mb	This comes in two modes: <ul style="list-style-type: none"> Standard resource mode (Production) allocates 128 GB per daemon Low resource mode allocates 48 GB per daemon For more information, see Resource planning .
Daemon Task Memory	llap_heap_size	Preconfigured to 48 GB in Standard resource mode and 16 GB in Low resource mode.
Daemon Cache Memory	Calculated difference between total memory allocation - (headroom + llap_heap_size)	Calculated value of approximately 70 GB. Total memory (128 GB) - Headroom and task memory (48 GB).
Headroom	Refers to an LLAP daemon node's memory used for non-task and non-cache requirements.	Configuration to specify number of available coordinators that trigger auto-scaling.

Related Information[Installing on premises Data Services](#)[Apache Tez processing of Hive jobs](#)[Planning a Virtual Warehouse instance](#)[Hive auto-scaling on Cloudera Data Warehouse on premises](#)

Migrate from Cloudera Base on premises (Hive on Tez) to Cloudera Data Warehouse (LLAP)

If you are on the Cloudera Base on premises platform and running your Hive workloads in the Tez container mode (Hive on Tez), learn how you can migrate to Cloudera Data Warehouse on premises and run Hive workloads on LLAP mode. LLAP offers ETL performance and scalability for complex data warehousing jobs.

Such a move can meet or exceed your customer satisfaction requirements, or cut the expense of running your jobs, or both. You set up automatic scaling up of compute resources when needed, or shutting down when not needed.

LLAP along with the query isolation feature is best suited for data-intensive queries, such as ETL queries, and require auto-scaling based on the total scan size of the query.

You perform the following high-level tasks to migrate from Cloudera Base on premises (Hive on Tez) to Cloudera Data Warehouse (LLAP):

1. Install Cloudera Base on premises.
2. Migrate your Hive workloads to Cloudera Data Warehouse.

3. Perform the post-migration tasks described in Apache Tez processing of Hive jobs.

Hive on Tez in Cloudera Base on premises vs LLAP in Cloudera Data Warehouse

The direct correlation between YARN on Cloudera on premises and Cloudera Data Warehouse on premises is the YARN queue. Every time you run a Hive query in Cloudera, Apache Tez requests a container from YARN. Tez jobs on YARN allocate containers based on the Tez configurations. Containers used to run queries consume a finite amount of memory but rarely use all that memory.

For example, consider a job that runs in 10 containers with each container using 4 GB of memory. This translates to 40 GB of memory used by the job. Now, if the actual memory used by the job is only 75% of 40 GB, it results in an overallocation of 10 GB of memory.

In comparison, task containers in Cloudera Data Warehouse are allocated 50% faster than the containers on YARN. Since the containers are part of a larger Java Virtual Machine (JVM), the memory is already allocated for the task and the memory is shared with other task containers in the larger JVM. This ensures that there is no overallocation of memory for each task to address the potential outlier memory requirement.

The improvements that a Cloudera Data Warehouse Virtual Warehouse instance has for task allocation as compared to YARN, translates to as much as 50% fewer resources (memory/cores (executors)) that are required to run the same job. A queue in YARN is 100% allocated with no elasticity whereas a Virtual Warehouse instance size with 50% of that capacity should be enough to start with.

If the YARN queue only reaches capacity during certain times of the day, then the Virtual Warehouse instance can just be a fraction of the total queue size if the instance is configured with concurrency auto-scaling.

If the jobs running on the Virtual Warehouse instance are unpredictable or are required only during certain times of the day, consider disabling AutoSuspend on the instance so that resources are given back when not used.

Some of the other advantages of Cloudera Data Warehouse (LLAP) over Hive on TEZ (YARN) are:

- Data Caching
- Faster task startup times
- Established sessions; no latency establishing a session

Related Information

[Installing on premises Data Services](#)

[Apache Tez processing of Hive jobs](#)

[Planning a Virtual Warehouse instance](#)

[Hive auto-scaling on Cloudera Data Warehouse on premises](#)

[Configuring AutoSuspend](#)

Migrating Hive workloads to ACID

An Apache Hive transactional table is also known as a Hive ACID table. The terms transactional and ACID are interchangeable. Data from transactions involving money, especially, but also other transactions, require databases that meet ACID requirements. You learn what ACID means and some features available if you use ACID tables.

This documentation describes Hive 3 under default conditions. All managed tables are ACID tables.

[ACID is an acronym](#) that describes database properties as follows:

A = Atomic: Changes to the database occur all at once or not at all.

C = Consistent: Before and after system failures, results of inserts and deletes are the same.

I = Isolated: A read operation is not affected by changes to the database that occur during the operation.

D = Durable: Successful transactions occur regardless of system failure.

Atomic means simultaneous operations do not interfere with each other. Once done, the result of an atomic operation is available to others. Transactions are consistent between operations. An operation is isolated in its own workspace and durable, meaning after being committed, data remains and survives any failure.

If the data about your transactions does not have to be atomic, consistent, isolated, and durable, you can configure Hive to create managed tables that are not ACID.

You get a number of capabilities described in the following sections when you use ACID tables.

Query results cache and data cache

Results of ACID tables that Hive manages on the file system and in HMS can be cached. To reliably deal with data caching and [query results caching](#), Hive needs to check that nothing is changed. If something changed, Hive invalidates those items to read more data. Hive must have complete control over the tables to perform the checks associated with caching.

System management of ACID tables

ACID tables, which are transactional tables, are managed by Hive. [Hive tightly controls data storage and access](#). By default, direct file system access to the managed Hive warehouse files is not allowed. Hive keeps you out of the warehouse. You do not need to access the Hive files on the file system. Access to the file system subverts security and governance on Hive tables beyond just reading the columns. The [data masking in columns and row filtering](#) are compromised if you have access to the files. Exceptions: <https://docs.cloudera.com/cdp-private-cloud-base/7.3.1/security-ranger-rms-configuring-and-using/topics/security-ranger-rms-introduction.html> policies are projected down to the file system level to handle Spark and [Hive Warehouse Connector](#) (HWC) Direct Reader access.

Hive stops external operations from changing managed files to prevent data integrity and corruption problems.

Compaction

Managed tables go through a . You can make inserts into table repeatedly and the ACID compaction system helps clean up inserts. The system compacts the changes, reducing your footprint of [small files \(delta files\)](#). You cannot, however, batch micro inserts, such as making 1000 inserts at one time. This creates problems with the system. Hive is a batch system, not intended to be handling online transaction processing (OLTP).

Hive statistics

Managed tables generates and manages [table statistics](#) for ACID tables. This is not necessarily true for other tables.

Governance of Hive tables

Hive tables are managed, audited, and have protected columns. You can protect the data level using [masking and row level filtering](#).

Advanced features

Advanced features, in addition to ACID, include materialized views, the query cache, and automatic statistics generation. You can update or delete data from ACID tables. Hive 3 addresses small files different from Hive 2.

For more information, see the Hive product documentation on the Cloudera web site:

- [Materialized views](#)
- [Query results cache](#)
- [Automatically collected statistics](#)
- Mutability
 - [Updates allowed](#)
 - [Deletes allowed](#)
- [Handling small files](#)

Tables in Hive 1 and 2 vs. Hive 3

Ownership, specifying the location of Hive tables, and the default location of Hive tables have changed in Hive 3. You gain an understanding of some restrictions and limitations of Hive 3 in Cloudera under default configurations. You need to understand how this change impacts your workflow.

In Hive 1 and 2, the owner of a managed table data is the table itself. If you drop the table you drop the data. In Hive 3, the system user hive [typically owns the managed table data](#). Exceptions include Hive 3 Streaming in which the streaming user owns the data. Hive performs [compaction](#) of the files. [Deltas](#) and the data location is controlled by Hive.

In the default Hive 3 in Cloudera, you typically [cannot specify a location](#) in a CREATE TABLE statement. Hive finds the location of the table location by first looking in the metastore default warehouse directory, the managed warehouse directory. The [default location of Hive tables](#) has changed in Hive 3. The managed warehouse is the default location of ACID files. There is also a location in the warehouse for external files. You can change a Hive property to change the warehouse directory, and if your HiveServer (HS2) is connected to the Hive metastore (HMS), the new locations take effect. You can change the location of managed tables and the location of external tables for a specific database. New tables created in that database use the new location.

In Hive 3, all managed tables are transactional (ACID). There is a tight relationship between the metadata and data. Hive keeps track of file system changes as well as metastore changes to satisfy features, such as column masking for fine-grained access restrictions.

Compatible storage formats

You need to know the recommended formats for Hive ACID tables, and how you can access tables from Spark. The HMS translation layer checks the capabilities of a Hive client that tries to access Hive and returns an error message designed to help you resolve an access problem.

For Hive ACID, ORC is the [recommended native storage format](#). You can insert, append, update, delete data in ORC format, but Hive ACID is not restricted to ORC. You can do insert/append operations to files in most other formats, such as text, Parquet, and AVRO. For more information about ORC, see [ORC file format](#) and [Advanced ORC properties](#) in Hive Performance Tuning.

Spark is not natively compatible with ACID tables. You need to use [Hive Warehouse Connector](#) (HWC) to read Hive ACID tables from the Hive metastore. There are two modes for HWC: [JDBC mode](#) and [Direct Reader mode](#). The [HMS translation layer](#) prevents Spark from accessing Hive tables.

The HMS translation layer checks each client connection to determine the capabilities of the client. For example, HMS checks whether or not the client supports ORC and transactional tables. When a Spark client talks to the metastore, it can bypass HiveServer (HS2). Some operations, such as Direct Reader and Hive Streaming, [go to Hive directly through HMS](#) and reveal its capabilities to the HMS translation layer. If Spark does not have a connection to HWC associated, when the Spark user tries to get information about the ACID table, the query fails. The HMS translation layer determines that Spark without HWC does not have the required capabilities to access ACID tables, and gives you an error.

Table design considerations

You need to understand the old way of designing tables and how bring some table design habits to ACID cause problems. Solutions come later.

Managing data ingestion frequency

Things to consider when designing a table with Hive include ingest frequency. Data ingestion has been the controlling factor in table designs on legacy platforms. ACID tables provide an opportunity to solve problems in legacy table designs.

On legacy platforms, you see a tendency to over-partition tables, not to gain an advantage from a consumer standpoint, but for a physical demarkation point for the ingest process. If anything goes wrong, you delete the partition representing the data for that ingest, and just move on.

A table design based on ingest frequency, which is the number of ingest events, can lead to an abnormal, inefficient number of small files. Each ingest event creates a new series of files. Historically, to overcome this, you used excessive partitions to define and mitigate problems from numerous appends to a physical location (base or partition).

Append operations on non-ACID tables create a small file problem.

Ingesting tables with numerous partitions, excessive append operations, and numerous READ operations result in the following problems:

- Poor consumer performance
- Increase compute requirements
- File system management stress

An ACID table helps prevent issues by:

- Compacting append operations, building better file sizes through compactions.
- Building and maintaining statistics
- Atomic operations

When you cannot alter data sources, design your workflow as follows:

- Use an ingest table to sweep to an ACID table (append).
- Execute READs on ACID tables.
- When partitions make sense, design them for the consumer, not the ingest pattern.

Streaming Data Sources:

- Hive streaming (to an ACID table)

Building high-level partitions for ACID tables

With compactions and CRUD capabilities, you need to design tables for the consumer.

Partitioning that works fine for non-ACID tables, such as YYYY-MM-DD-HH, which represents 8760 partitions a year, is not recommended for partitioning ACID tables. Build a higher-level partition, such as YYYY-MM, which represents 12 partitions a year.

Let compactions manage and optimize the appends. Optimize by using larger files per partition. ORC file types with columnar formatting, statistics, headers, footers, and Bloom Filters help optimize what Hive must scan, reducing I/O.

Tracking batches

To track batch Hive processes:

- Add a field to the table with the batch id.
- Use ORC CRUD functionality to remove and replay a batch instead of dropping the partition.

Hive ingest patterns introduction

Understanding what does not work when designing Hive tables helps you understand recommended patterns discussed. You can avoid potential performance issues, and perhaps data loss.

Operations on non-ACID tables create a small file problem. Appending small, non-ACID files to the same partition or table generally prevents consolidation of files inside the partition directory. Consolidation happens only after an INSERT OVERWRITE to the table or partition.

Using classic INSERT OVERWRITE methods can lead to data loss. Data not picked up at beginning of INSERT OVERWRITE can be lost. INSERT OVERWRITES are not atomic, so for a time during that operation, there will be no data in the table when HMS is processing the data.

By contrast, ACID tables are consolidated through the compaction process. When you insert data into a non-ACID table before writing results to target partition or table, Hive tries to write to the file as if the file were new, and empty. An object storage failure, such as an HDFS failure, occurs as the file already exists because of a previous insert. Hive renames the file to copy 1 and tries again. If another failure occurs, which it will because copy 1 exists in that dir, it renames the file to copy 2.

An ACID anti-pattern is doing 1400 inserts a day to a relatively small table. Hive needs to interact with the NameNode 1400 times, or more, just to insert a single file into a table. It must fail in each of the 1400 iterations before finding a number that works. In addition to a small file problem, the thrashing activity overwhelms the NameNode.

Creating a partition per ingest cycle is another classic pattern for managing batches. The pattern leads to the following problems:

- Extremely poor performance on the consumer side
- Many small files
- Enormous pressure on the metastore and filesystem
- More compute to handle queries

Avoid ingesting tables having numerous partitions and heavy append operations. Consolidate files. Doing so saves time and resources, and relieves stress on the NameNode. Running compaction of ACID tables achieves this consolidation and prevents these problems. Hive also collects ACID table statistics. All ACID operations are atomic.

When your data source cannot be altered, for example when the frequency of ingesting data is high, you need to keep the source data intact. You can use a sweep operation, or ingest table, to sweep data into an acid table. The ingest table can be an external table populated by NiFi, for example. NiFi reads data from Kafka and writes files to HDFS. Using a sweep table you append data to the ACID table on a less frequent basis. This technique helps you manage the operation. Read operations from consumer side go through the ACID table, which can be consumed efficiently.

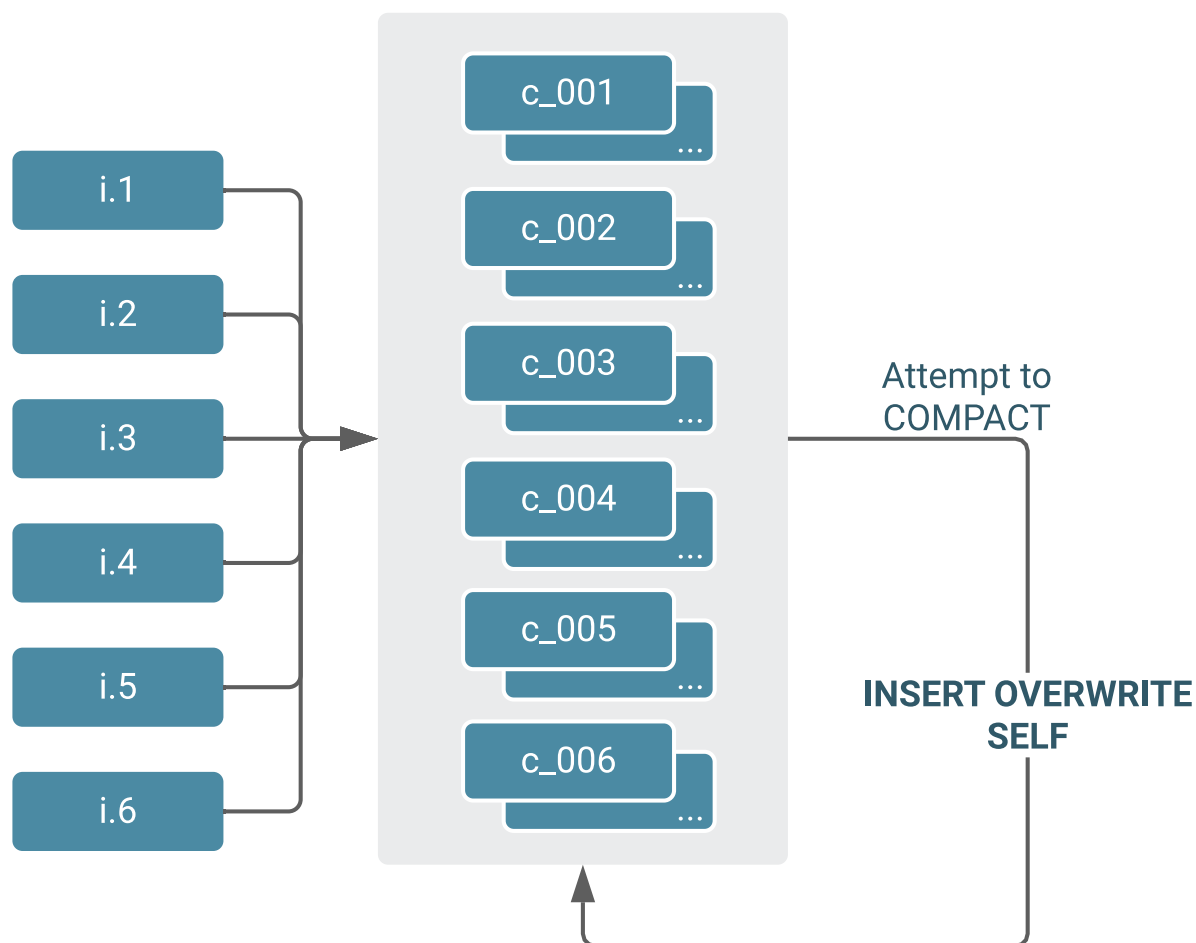
When partitions make sense, design the partitions for the consumer, not for the ingest pattern to the final target acid table.

Classic ingest patterns

You need to move away from ingest patterns commonly used for Hive 1 and 2 that are consumer centric to avoid performance problems on the consumer side.

The following diagram shows the classic partition that addresses the ingest pipeline instead of the consumer pipeline. Multiple appends to the table or partition that create small files are minimally addressed with INSERT OVERWRITE. This mitigation is not atomic.

Classic Pattern 1



This classic pattern shows repeatedly inserts: creating copy 1, copy 2, copy 3, copy 4. When you get to the sixth insert, Hive tries to write the original file, but fails because it already exists. The write attempt will fail 6 times before the insert succeeds.

This pattern can be summarized as follows:

- Brute force.
 - Small files
 - Poor performance
- Counter by:
 - Overwrite compaction job

Problems:

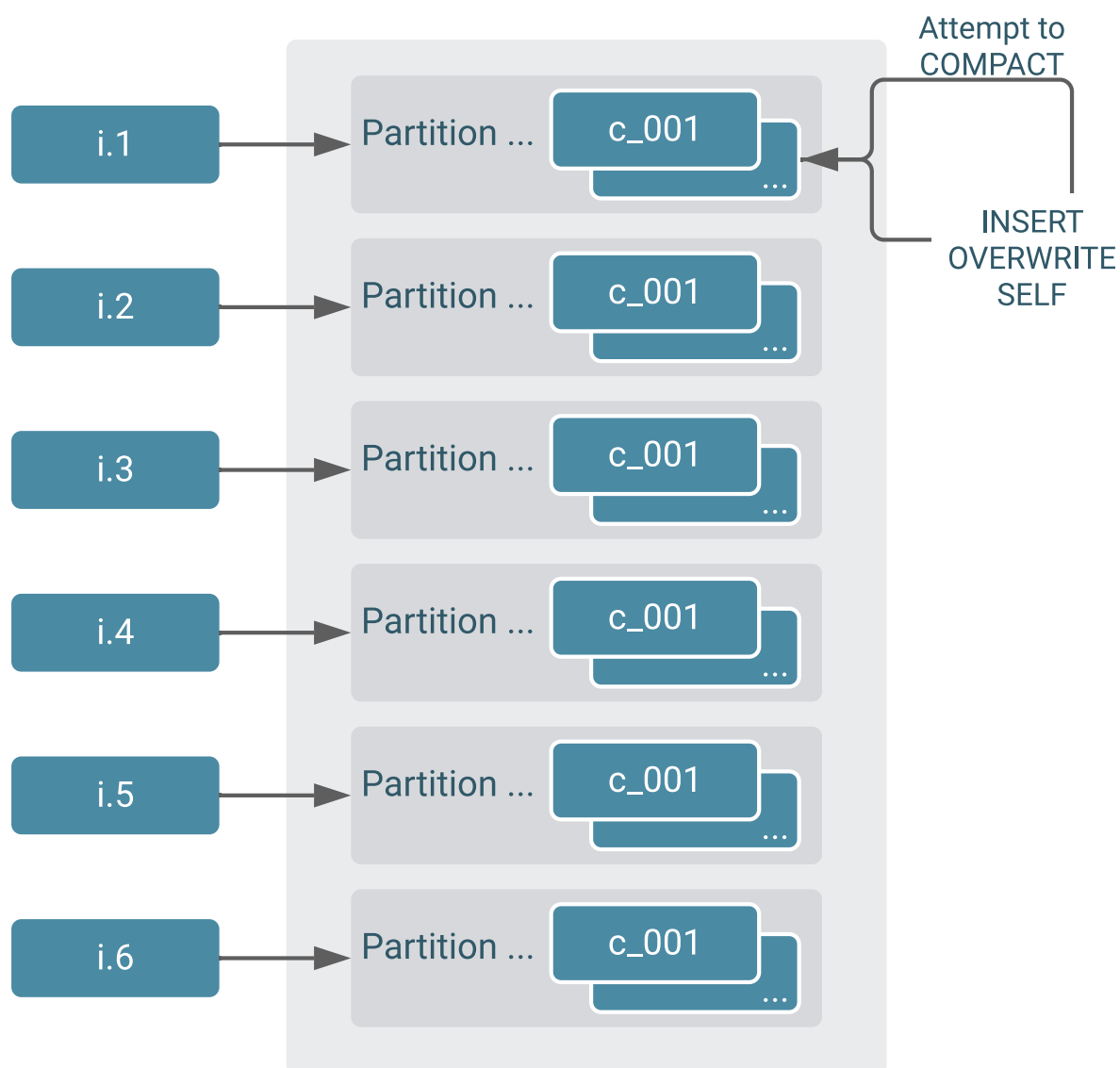
- Not atomic
- Data if ingest events are not stopped

Classic pattern using partitions

When using Hive 1 or 2, creating a partition per ingest cycle helps you manage batches. This pattern results in extremely poor performance on the consumer side for the following reasons:

- Generates many small files
- Pressures the metastore and file system
- Requires more compute to handle queries

Classic Pattern 2



Classic pattern 2 is similar to classic pattern 1, but adds a physical abstraction: partitioning.

Classic pattern 2 adds the pressure of partition management at the Hive metastore (HMS) level. Creating partitions per cycle, for example on day, month, year, and hour was a convenient way to manage batches for Hive 1 and 2. In Hive 3, partition ACID tables at a higher level. Instead of partitioning by day, for example, partition by month. Consider the following things:

- How frequently you ingest the data.
- How often you update the data.
- The size of the data.

A daily partition that yields 4MB of data makes sense on the ingest side, but causes problems on the consumer side. In this case, it makes sense to change the partitioning from daily to monthly. To improve the yield, partition 4MB x 30 days of data a day to yields 120MG of data. The number of small files is reduced. Hive compacts the files. Queries hit a higher density of records inside the same size ORC file.

Small partitions lead to HMS performance pressures and other problems, especially during heavy queries to the same table. Heavy reads greatly increase the load on Hive metastore (HMS) to build the partition list. If partition pruning

does not occur, performance degrades. Build partitions of ACID tables on a level appropriate for your data volume. You need a huge amount of data to justify partitioning by month, day, and hour, which represents 8700 partitions per year.

Partitioning by month reduces the number of compaction operations and optimizes append operations.

ACID ingest patterns

Understanding Hive ACID ingest patterns helps you adopt one that fits best. You gain an understanding of how to build a pipeline that keeps the original data and builds or updates a more efficient table for recurring READ operations.

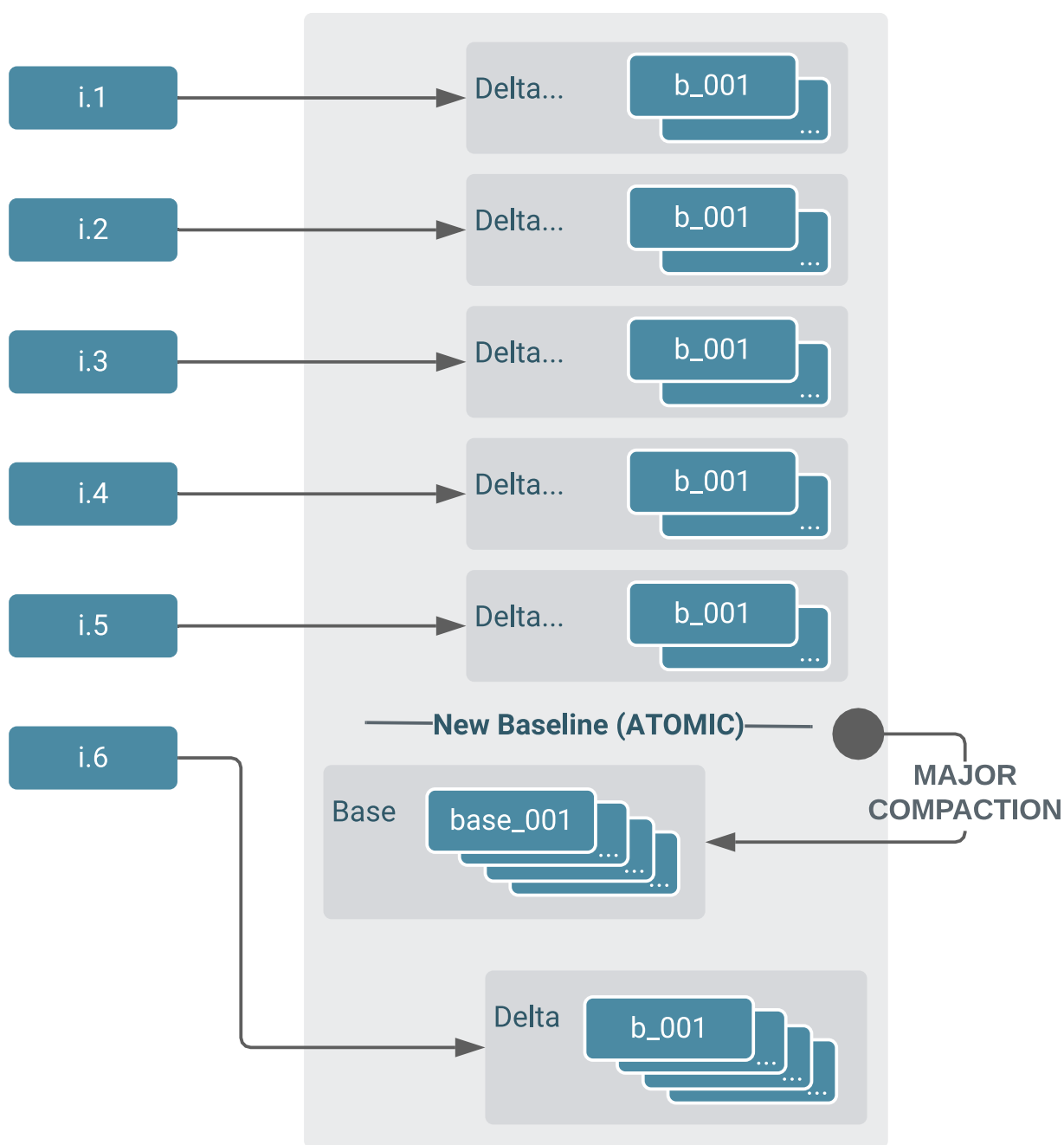
Although Hive handles compactions, micro-batching appends, and Hive streaming writes, you still have to avoid inserting small records into ACID tables. Using ACID does not correct a bad ingest design. If you perform micro-inserts and create many delta directories, at some point the compaction system, and other components, such as NameNodes, have to reconcile the delta files. Eventually, compaction consolidates files, but if you have hundreds of these delta files before compaction even starts, Hive needs to work hard in the background. Heavy compute resources and metastore resources are needed.

The data you ingest into ACID tables using the following pattern must be of a reasonable size.

ACID pattern 1

ACID pattern 1 characteristics are:

- Handles compactions in the background, but you still have to understand the impact of deltas
- Performs well for micro batch appends and Hive streaming
- Works best when you partition on business need, not ingestion
- Frequently partitioned to optimize file size and access (pruning)
- Supports adding a batch-id field to record ingest events
- Not designed for online transaction processing (OLTP)



Consider how quickly you need to access the data. If you need immediate access to data, look at how many queries your organization actually issues to access data received within the last 5 minutes, for example. You might realize that rarely do you need access your data so quickly, but if not, consider other technologies, such as Impala with Kudu or HBase.

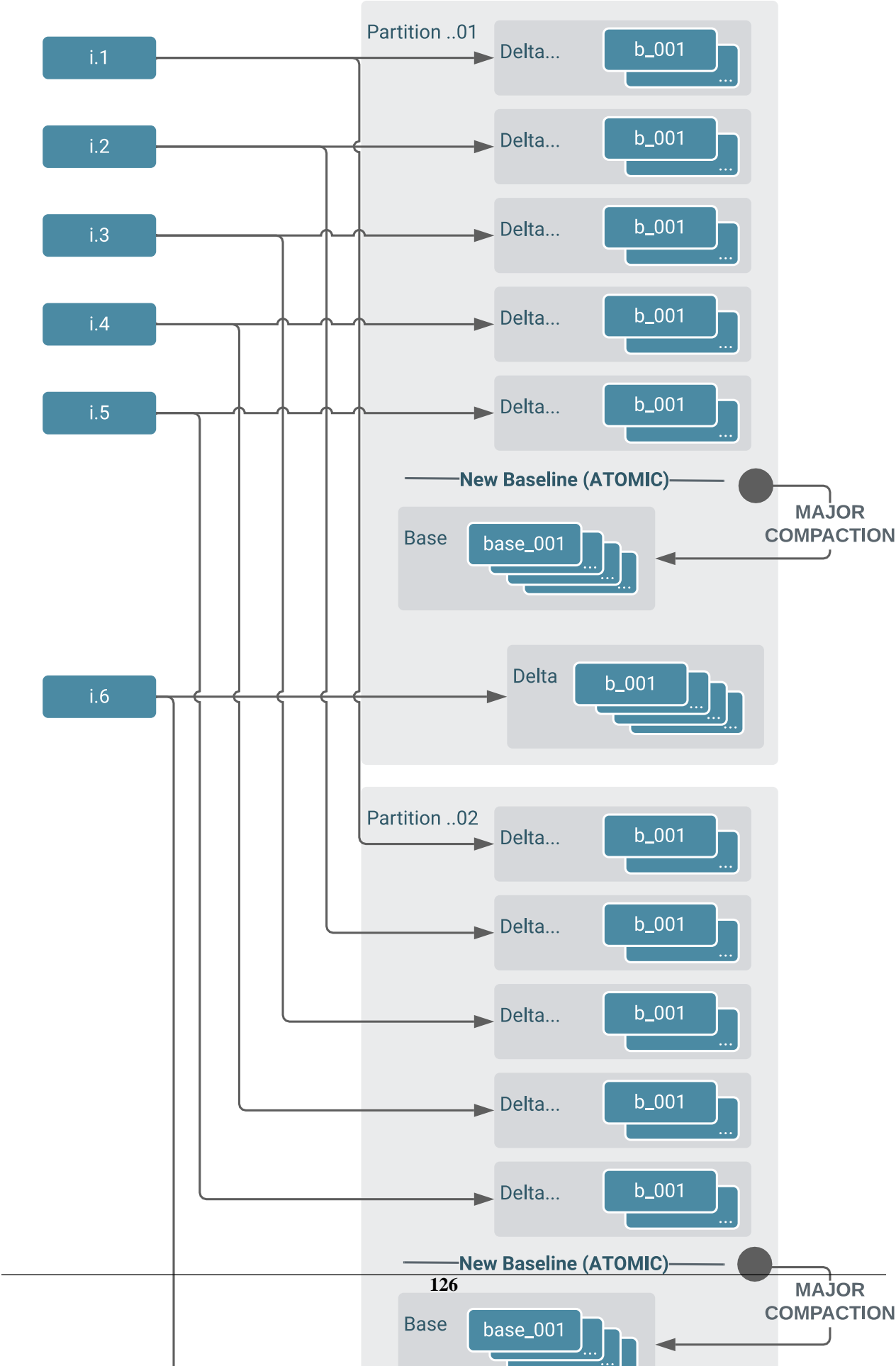
If you need to track batch operations, for example by associating a batch ID with every record, add a second-level partition element. Add a batch field to the table with the batch ID. You can perform delete operations against ACID tables to remove a batch and replay it. ORC and CRUD functionality repair that table based on a replay or removal of an insert.

Hive does not satisfy OLTP requirements.

ACID pattern 2

ACID pattern 2 has the following characteristics:

- Designed to be business-, not ingest-centric.
- Supports highly granular partitions, for example YY-MM-DD-HH vs YY-MM.
- Achieves efficient content size per partition to reduce file counts.



Beware of [dynamic partitions](#) and avoid cross partition distribution of data. If your design requires Hive to cross partitions unnecessarily when you insert data into a table, collapse year-month-day-hour partitions down to year-month if possible.

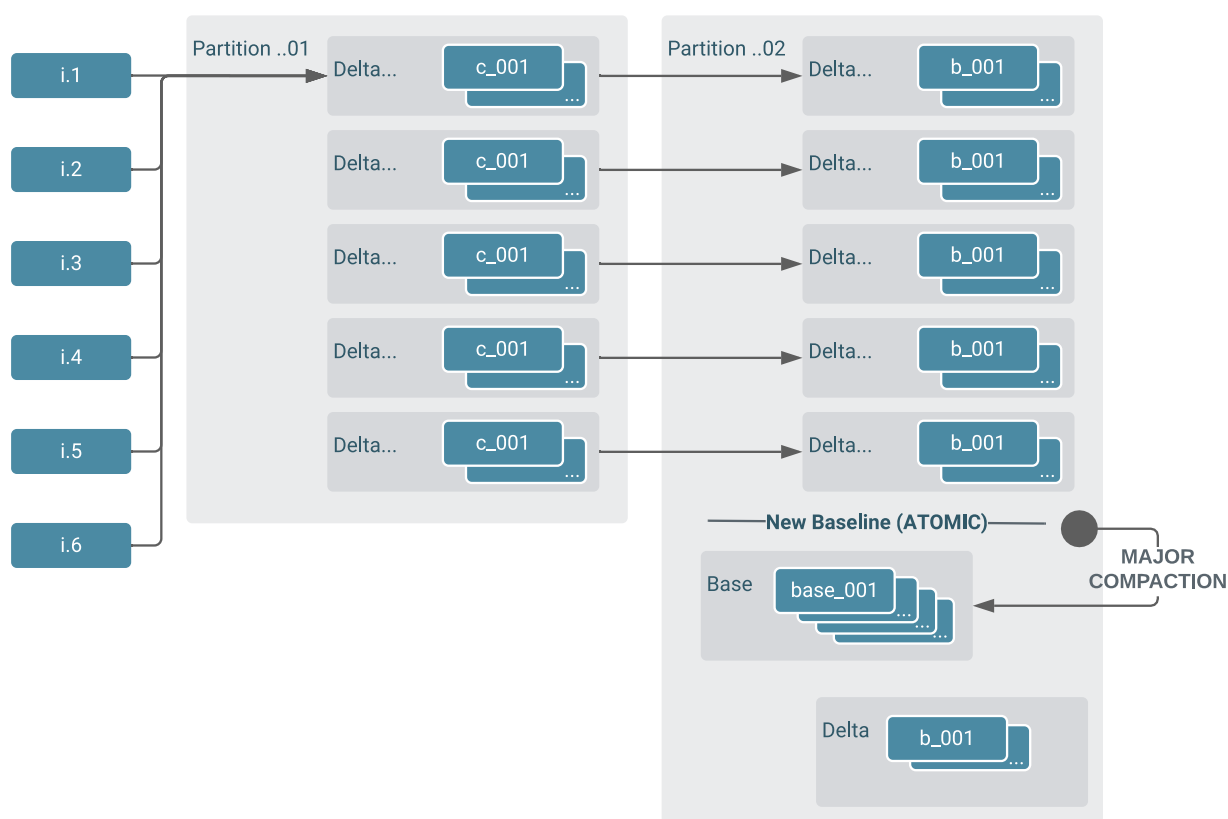
ACID pattern 3

To build a pipeline that keeps the original data and builds or updates a more efficient table for recurring READ operations, use the ACID pattern 3.

ACID pattern 3 represents the sweep process. This pattern keeps track of historical changes. You can lose information that has business value if you do not keep track the original transactional elements. Consider having an ingest table with original values and also a change data capture (CDC) table. For example, if you have 2 million customers making thousands of changes a day to an ACID table target, you lose all the thrashing that might have happened if you do not capture changes.

Using a sweep process not only consolidates files to alleviate ACID performance problems, but also supports data change analytics. If a consumer changes their address frequently, say 50 times a day, perhaps fraud is indicated. The cost of space you need for historical data is often worth the expense.

A portion of the sweep pattern, shown below, looks similar to the classic ingest pattern. You use a non-acid table with partitions. Instead of inserting data into a non-ACID table every 15 minutes, for example, you instead sweep data from the ingest table into the ACID table every hour or two. You use the ACID table as your consumer table, which has collapsed partitions. Hive performs compaction on the ACID table.



Another approach is to use an ACID table as the staging place for ingesting data or other data pipelines for writing and aggregating data. Turn off auto-compaction, or raise thresholds, to enjoy transaction isolation of your streaming with no overhead.

In summary, use ACID pattern 3 as follows:

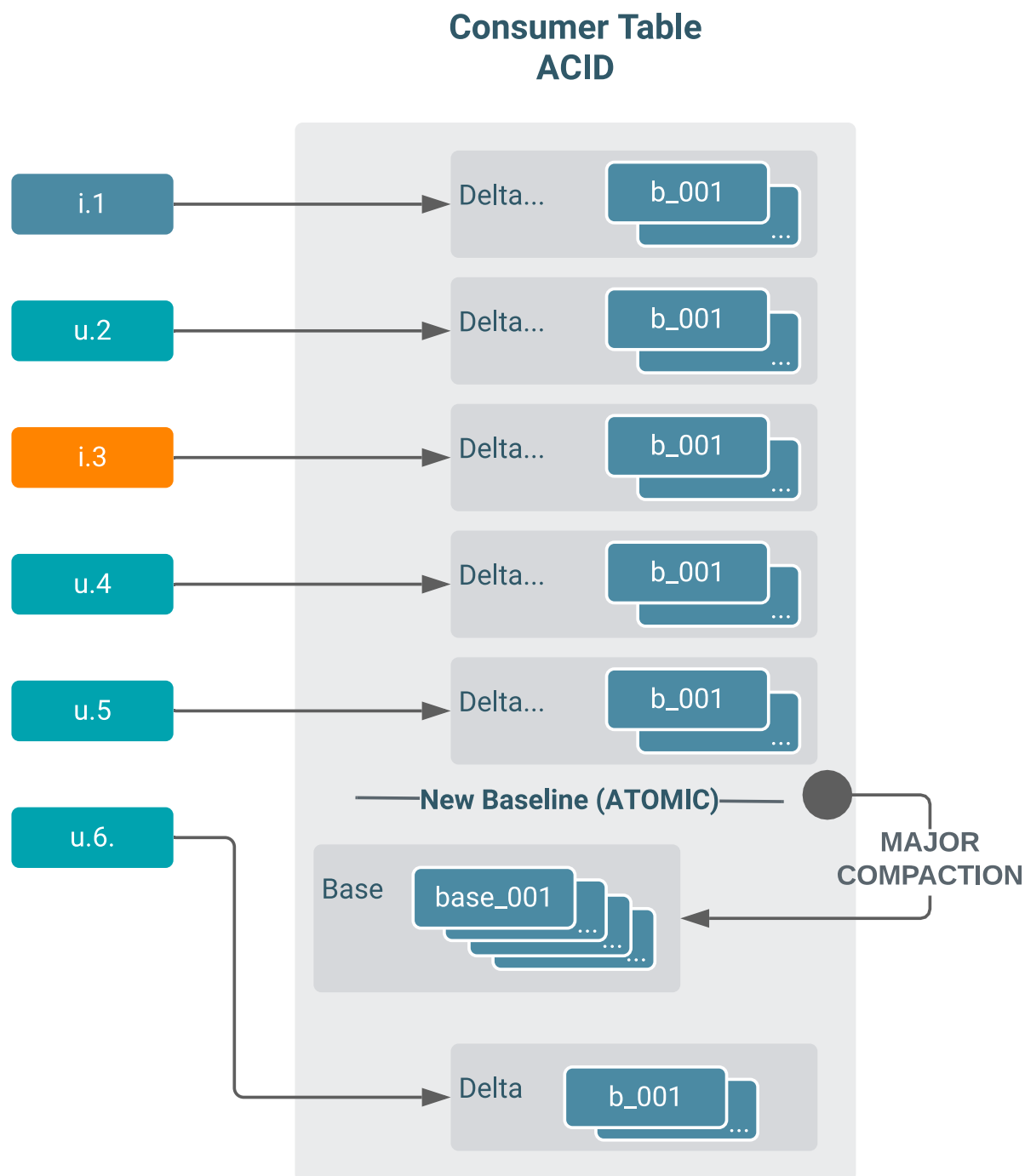
- Use a non-ACID table to stage micro batches to a final ACID table.

- Use more aggregate partition strategy (if required) on the final table (YYMM vs. YYMMDD_HH).
- Allow ACID to undergo compaction.
- Direct consumers to the final ACID table for better performance and less system resource overhead.
- Include a batch ID in the schema to support rollback.

ACID CDC pattern

The ACID CDC pattern has the following characteristics:

- Used for the initial seed of new records.
- Used for follow-up updates and inserts.
- Used for follow-up deletes.
- ACID deltas track updates and deletes, rolled up in SELECT.
- Relys on major compaction to reconcile updates and deletes.



For changing dimension tables, the ACID CDC pattern supports a large partition, late arriving data, or perhaps no partition at all. Using the ACID CDC pattern, you might have just a consumer table. Inserts occur, and then updates to the inserts. All changes are captured as delta records. A major compaction reconciles inserts and updates to give you a new base. The next update creates another delta in this case.

When you insert into an ACID table, you must deduplicate the records before making insertions, or deduplicate the records before updates. You cannot skip the deduplication process because there is no enforceable constraint primary key that takes care of eliminating duplicate copies of data. If you put a record in a transactional table three times before it is pushed into an ACID table, the record comes into Hive three times: first as an insert, next as an update, and finally as another update before you push your changes into the ACID table through the merge process. You must

reconcile inserts and updates before the merge to make sure you get only one operation; otherwise, you get duplicate records.

Handling government regulations in ACID tables

Your decision to use ACID tables, or not, might depend on your need to conform to government regulations. With consumers right to be forgotten growing around the globe, you need some way to purge these consumers from your datasets. ACID supports features to meet these regulations.

Meeting the following government requirements is often subject to interpretation, but involves deleting records of consumers who want to be forgotten on a web site.

Europe - [GDPR](#)

California - [California Consumer Privacy Act 2018](#)

Before you propagate Hive data, you can check for requests to be forgotten and comply with requests. ACID supports deleting records. ACID completely removes data after a major compaction.

ACID tables support delete operations on datasets using MERGE. Operations are atomic, and are built into an ACID table.

Key concepts about ACID ingest patterns

There are many variations to the ACID ingest patterns discussed earlier. The main points are:

- Using an ACID ingest pattern, the system manages file compactions in an atomic manner.
- Do not force your ingest patterns on the consumer through artificial partition strategies.
- If you need to maintain the source, use the sweep pattern. Consumers access the final table.
- Minimize repeated reads of inefficient datasets.

The following enhanced policies help protect sensitive data at a column or row level versus giving consumers file access to all data:

- [Column masking and row filtering](#)
 - Removes sensitive data from unauthorized users.
 - Prevents derivative datasets and loss of control.
- [Geographic policies](#)
 - Adds control depending on where you are.

After giving a consumer access to the file system, there is no way to protect the sensitive parts of the data.

Treat managed, ACID tables as tables, not as files. Use column masking and row filtering to restrict access to sensitive data. Hive operations go through HiveServer (HS2). All policies are managed around the table context. All operations on managed tables run as the hive user on service, who is analogous to the oracle user who is the user accessing an Oracle system.

With delineation and enforcement of ACID tables, ACID tables become a single source of truth (SSOT). You can control file spillage, and provide theft protection by limiting user access. The file system audits are not always available for non-ACID tables in non-HDFS or ozone stores. If you have data in S3, for example, allow users go through Hive but do not give them access to S3, then you can obtain an audit trail.

Modified ACID table location

Hive 3 does [not allow you to specify a location](#) when creating a table. Hive 3 does not allow LOCATION declarations in the CREATE TABLE statement for the following reasons.

- Managed table file locations have stronger security requirements in Hive 3 than in Hive 1 and 2.

- Hive 3 ensures the location matches policies that manage the space.

If the default locations of the Hive warehouse do not suit your use case, for example, if you have multiple TDE zones or you like to control where you put things, you can override the warehouse location setting in the Hive metastore. You [set a MANAGEDLOCATION](#) database property, and then new tables you create are stored in this managed location. To see the managed location, you use a DESCRIBE DATABASE query.

You might need a custom Hive 3 warehouse directory location to meet the following requirements:

- Multi-tenant requirements
- Transparent data encryption (TDE) requirements

The [default warehouse location](#) is controlled by the hive.metastore.warehouse.dir property in the Hive metastore. Key points about the Hive 3 warehouse directory are:

- Set the MANAGEDLOCATION property of the database using CREATE TABLE OR ALTER TABLE.
- Check the property value using DESCRIBE DATABASE.
- Ensure access to your new file system location for the hive service user.
- For [transparent data encryption \(TDE\)](#):
 - Ensure that the hive service account has key access.
 - Use an alternate warehouse location.
 - Protects data at rest if the file should fall into the wrong hands.

Accessing Hive ACID tables from Spark

Spark interaction with ACID tables is not natively supported. When deciding if Spark users need access to ACID tables, or not, determine your data priorities and the risk of exposing the data. If your security concerns outweigh ease of access, ACID tables are a viable choice for use with Spark. You can protect parts of the database that Spark users access.

Spark users cannot just run sparksql code on ACID tables. For governance reasons, users must go through the [Hive Warehouse Connector \(HWC\)](#) and make code changes to access the non-native ACID tables. You need to consider the options for accessing the non-native tables and the impact of GDPR (General Data Protection Regulation) to determine if using ACID tables for Spark users works for you, or not.

Hive 3 on the Cloudera Data Platform [does not support storage-based authorization \(SBA\)](#). The default [Hive doas property](#) is false, and results in code changes. These changes are mandatory to secure your data. Using old patterns that required insert overwrites can cause data loss and slow Hive process. If you continue to use these pattern with ACID, Hive will be slow. Hive 3 operations are atomic and require backend housekeeping. When you use acid table,s you have to give up some part of the system to manage them with compaction. How much of your resources are needed for compaction depends on how extensively you use ACID tables.

ACID has limitations, such as writing a thousand INSERT statements containing one record. Such inserts cause problems. Hive ACID is not an online transaction processing (OLTP) system, but it can you can build a better and cleaner data pipeline with Hive ACID, and it can perform well.

Audits, lineage, and enforcement

By applying policies to Hive ACID tables your Hive database can be a single source of truth for users. File spillage, loss, and theft is protected by [restricting user access](#) to the file system database and tables using Ranger.

Views and materialized views

A [materialized view](#) is a database object that holds a query result you can use to speed up the execution of a query workload. Materialized views require ACID tables.

In the following example, you select a row ID, perform an aggregate function, and group by consumer:

```
SELECT consumer_id, agg_function FROM detail_table GROUP BY consumer_id;
```

The aggregate function is costly, depending on how the cost of scanning the table, how big the table is, and so on. The aggregation is calculated each and every time you run that query.

Suppose you use a view. VIEW wraps the SELECT syntax. The view reference replaces the select, and is compiled by Hive in the Directed Acyclic Graph (DAG). A view might appear easier to use and consume, but is no better, or is worse, performance-wise than a materialized view. There is only a development advantage in using a view over a materialized view.

Suppose you use a materialized view of Select with the same aggregations. The materialized view can be updated or maintained in the background on an occasional basis. If you run aggregates on a frequent basis, the materialized view might save significant time. You create a materialized view that matches the first aggregate function in the table. When a user issues a query that includes the aggregation against the table again, the Hive Cost Based Optimizer (CBO) checks that the materialized view is up-to-date with that aggregation, [rewrites the query](#) if out-of-date, and uses the previously generated results instead of reading the entire table again.

Advantages of materialized views are:

- The speed of repeated dashboard-like queries improve.
- Query results are pre-built.
- Queries are optimized by the CBO:
 - Query plans use materialized views if possible
 - Reduces data Processing
 - Quicker response times
 - Fewer resources required

Query results cache

If you use ACID tables, repeated queries can be resolved by the [query results cache](#). Query results caching works only on ACID tables because Hive needs to completely manage the tables to do the caching. Hive knows whether the ACID tables changed or not, so when you issue the query again, Hive just pulls the result without further computation. Hive doesn't need to run the query again. If you have a dashboard, perhaps the first dashboard that runs the query will cache the data for the next hour until the dashboard needs to be updated. User queries pull the results from the cache, and save all of that compute time and resources.

You can configure Hive to manage a certain amount of space for the query result cache.

The query results cache identifies an asymmetric tree of the query DAG identical to a query that was run before and stored on the cluster. No changes to tables occurs during query results caching.

Results are cached and presented without execution to the user for:

- Repetitious queries
- Queries that have not changed since results cache was built

Summary of Hive ACID considerations

Key things to keep in mind when moving to Hive transactional/ACID tables are:

- Hive is not an online transactional processing (OLTP).
- Micro-batching works best; avoid single inserts.
- [Setup the compactor](#).
- Scale the resources to compact.
- Resource requirements depend on:
 - Data volume
 - Frequency of data change
 - Number of tables/partitions to compact
 - The thresholds set to trigger compactions
 - The frequency of the compactor checks

One of your main considerations in whether or not to use a Hive ACID table is who is going to query the table. Hive has native support for ACID tables and CDP has a security system in place to protect the data.