Cloudera Runtime 7.2.11

Managing Apache Hive

Date published: 2019-08-21
Date modified: 2021-09-08

https://docs.cloudera.com/

Legal Notice

Contents

ACID operations in Cloudera Data Hub

Apache Hive supports ACID (atomicity, consistency, isolation, and durability) v2 transactions at the row level without any configuration. Knowing what this support entails helps you determine the table type you create.

By default, managed tables are ACID tables. You cannot disable ACID transactions on managed tables, but you can change the Hive default behavior to create external tables by default to mimic legacy releases. Application development and operations are simplified with strong transactional guarantees and simple semantics for SQL commands. You do not need to bucket ACID v2 tables, so maintenance is easier. With improvements in transactional semantics, advanced optimizations, such as materialized view rewrites and automatic query cache, are available. With these optimizations, you can deploy new Hive application types.

A Hive operation is atomic. The operation either succeeds completely or fails; it does not result in partial data. A Hive operation is also consistent: After an application performs an operation, the results are visible to the application in every subsequent operation. Hive operations are isolated. Your operations do not cause unexpected side effects for other users. Finally, a Hive operation is durable. A completed operation is preserved in the event of a failure.

Hive operations are atomic at the row level instead of the table or partition level. A Hive client can read from a partition at the same time another client adds rows to the partition. Transaction streaming rapidly inserts data into Hive tables and partitions.

Configuring partitions for transactions

You set a couple of parameters, to prevent or permit dynamic partitioning, that inserts, updates, or deletes data into partitions implicitly created on the table.

About this task

Configuring partitioning involves changing the following parameters to meet your needs:

- hive.exec.max.dynamic.partitions
- hive.exec.max.dynamic.partitions.pernode

You set hive.exec.dynamic.partition.mode to strict to prevent dynamic partitioning or to nonstrict (the default) to include INSERT, UPDATE, and DELETE statements in your transaction applications.

Procedure

1. In Cloudera Manager Clusters select the Hive service. Click Configuration, and search for hive-site.xml.
2. In HiveServer2 Advanced Configuration Snippet (Safety Valve) for hive-site.xml, click + and add the hive.exe c.dynamic.partition.mode property.
3. Set the value to nonstrict.
4. Save the changes and restart the Hive service.

Related Information

Hive Configuration Properties documentation on the Apache wiki

Options to monitor transactions

As a Hive administrator, you can view the list of all currently open and aborted transactions using the SHOW TRA NSACTIONS statement or by querying the TRANSACTIONS view within the SYS database.

The query statements display the following details about transactions:

- TXN_ID: Unique internal transaction ID
- STATE: Transaction state

- STARTED: Timestamp when the transaction was started
- LAST_HEARTBEAT: Timestamp of the latest heartbeat
- USER: Hive user who initiated the transaction
- HOST: Host machine or virtual machine where the transaction was initiated
- HEARTBEAT_COUNT: Total number of heartbeats
- TYPE: Transaction type
  - DEFAULT
  - REPL_CREATED
  - READ_ONLY
  - COMPACTION
- TC_DATABASE: Hive database name
- TC_TABLE: Table name
- TC_PARTITION: Partition name (if the table is partitioned)
- TC_OPERATION_TYPE:
  - SELECT
  - INSERT
  - UPDATE
  - COMPACT
- TC_WRITEID: Unique internal write ID

The following sections describe the various options that you can use to monitor transactions.

SHOW TRANSACTIONS

You can run the SHOW TRANSACTIONS statement to view details about all open and aborted transactions.

SHOW TRANSACTIONS;

This statement lists all the transactions and you cannot filter or limit the results as required. Alternatively, you can use the SYS database to query and filter specific transactions.

Querying the SYS database

You can query the TRANSACTIONS view within the SYS database to filter and view specific transactions.

For example, you can run the following query to view transactions in a particular state:

```
SELECT *
FROM SYS.TRANSACTIONS
WHERE STATE='aborted';
```

Important: Currently, you cannot retrieve information about old or aborted transactions from the SYS database that is preventing the Cleanup phase. This is because the MIN_HISTORY_LEVEL table is not exposed in the SYS database. As a workaround, you can run the following query directly on the backend database:

```
SELECT * from TXNS where txn_id = (
    SELECT MIN(res.id) FROM (
        SELECT ntxn_next id from next_txn_id
        UNION ALL
        SELECT MIN(mhl_min_open_txnid) id FROM min_history_level
        UNION ALL
        SELECT MIN(txn_id) id FROM txns WHERE txn_state = 'a'
) res);
```

Note: Similarly, you can also query the INFORMATION_SCHEMA database for details about transactions. You must use the Ranger service and set up access policies for Hive users on this database to make it accessible.

Options to monitor transaction locks

As a Hive administrator, you can view information about locks on a table, partition, or schema that are created as a result of transactions. You can either use the SHOW LOCKS statement or query the LOCKS view within the SYS database to view transaction locks.

Hive transactions, enabled by default, disable ZooKeeper locking. DbLockManager stores and manages all transaction lock information in the Hive Metastore. Heartbeats are sent regularly from lock holders and transaction initiators to the Hive metastore to prevent stale locks and transactions. The lock or transaction is aborted if the metastore does not receive a heartbeat within the amount of time specified by the hive.txn.timeout configuration property.

The query statements display the following details about transaction locks unless ZooKeeper or in-memory lock managers are used:

- LOCK_EXT_ID: Unique internal ID of a lock request that might be requesting multiple lock entries on several resources (tables or partitions).
- LOCK_INT_ID: Unique internal ID of a lock entry that has been requested by a LOCK_EXT_ID
- TXNID: Transaction ID associated with the lock, if one exists
- DB: Hive database name
- TABLE: Table name
- PARTITION: Partition name (if the table is partitioned)
- LOCK_STATE:
  - acquired: transaction initiator holds the lock
  - waiting: transaction initiator is waiting for the lock
  - aborted: the lock has timed out but has not yet been cleaned
- LOCK_TYPE:
  - exclusive: lock cannot be shared. No one else can hold the lock at the same time.
  - shared_read: any number of other shared_read locks can lock the same resource at the same time
  - shared_write: any number of shared_read locks can lock the same resource at the same time, but no other shared_write locks are allowed
- LAST_HEARTBEAT: Last time the holder of this lock sent a heartbeat
- ACQUIRED_AT: Time when the lock was acquired, if it has been acquired
- USER: Hive user who requested the lock
- HOST: Host machine or virtual machine on which the Hive user is running a Hive client
- HEARTBEAT_COUNT: Total number of heartbeats
- BLOCKEDBY_EXT_ID: ID of the lock (LOCK_EXT_ID) causing current lock to be in “waiting” mode, if the lock is in this mode
- BLOCKEDBY_INT_ID: ID of the lock (LOCK_INT_ID) causing current lock to be in “waiting” mode, if the lock is in this mode

The following sections describe the various options that you can use to monitor transaction locks.

SHOW LOCKS

You can run the SHOW LOCKS statement to view details about all transaction locks. Ensure that transactions are enabled

SHOW LOCKS;

The following examples illustrate some sample queries that you can run:

Query to check table locks

SHOW LOCKS mytable EXTENDED;

Query to check partition locks

SHOW LOCKS mytable PARTITION(ds='2018-05-01', hr='12') EXTENDED;

Query to check schema locks

SHOW LOCKS SCHEMA mydatabase;

The SHOW LOCKS SCHEMA cannot be used with ZooKeeper or in-memory lock managers.

The SHOW LOCKS statement lists all the transaction locks and you cannot filter or limit the results as required. Alternatively, you can use the SYS database to query and filter specific locks.

Querying the SYS database

You can query the LOCKS view within the SYS database to filter and view specific locks.

The following examples illustrate how you can run queries on the SYS.LOCKS view to monitor transaction locks:

Query to view locks requested on a particular resource (table or partition)

```
SELECT *
FROM SYS.LOCKS
WHERE db='default'
AND table='tab_acid';
```

Query to view list of acquired locks

```
SELECT *
 FROM SYS.LOCKS
 WHERE lock_state='acquired';
```

Query to view blocking transactions that are preventing locks requested by a user- defined transaction, from being acquired

```
SELECT *
FROM SYS.TRANSACTIONS
WHERE txn_id IN (
  SELECT txnid
  FROM SYS.LOCKS
    INNER JOIN (
  SELECT blockedby_ext_id, blockedby_int_id
  FROM SYS.LOCKS
    WHERE txnid=4534) b
  ON lock_ext_id = b.blockedby_ext_id
    AND lock_int_id = b.blockedby_int_id
);
```

Note: Similarly, you can also query the INFORMATION_SCHEMA database for details about transaction locks. You must use the Ranger service and set up access policies for Hive users on this database to make it accessible.

Related Information

Apache wiki transaction configuration documentation

Data compaction

As administrator, you need to manage compaction of delta files that accumulate during data ingestion. Compaction is a process that performs critical cleanup of files.

Hive creates a set of delta files for each transaction that alters a table or partition. By default, compaction of delta and base files occurs at regular intervals. Compactions occur in the background without affecting concurrent reads and writes.

There are two types of compaction:

- Minor

  Rewrites a set of delta files to a single delta file for a bucket.
- Major

  Rewrites one or more delta files and the base file as a new base file for a bucket.

  Carefully consider the need for a major compaction as this process can consume significant system resources and take a long time. Base and delta files for a table or partition are compacted.

  You can configure automatic compactions or do manual compactions. Start a major compaction during periods of low traffic. You use an ALTER TABLE statement to start compaction manually. A manual compaction either returns the accepted compaction request ID or shows the ID (and current state) of a compaction request for the very same target. The request is stored in the COMPACTION_QUEUE table.

The compactor initiator must run on only one HMS instance at a time.

Related Information

Apache Wiki transactions and compaction documentation

Compaction tasks

Compaction in Hive goes hand-in-hand with Hive ACID. Compaction is not, however, necessarily required for Hive ACID. You need to understand when you want, or do not want, compaction to occur.

If you confine your ACID operations tables to full reloads and some delta merges, the performance is steady. As tables are always rewritten (dropped and recreated), there is no need for compaction. Consider disabling compaction by

Compaction occurs for the following reasons:

- You explicitly trigger compaction on a table or partition.
- Automatic compaction finds something to compact.

You run an ALTER TABLE statement to start explicit compaction. Automatic compaction happens without your intervention when Hive periodically crawls through the transaction information, finds tables/partitions affected and those fit for the pre-defined conditions, and marks them for compaction. Conditions are based on the number of deltas, amount of change, and so on.

Compaction of sorted tables is not supported.

In a data pipeline, creating staging or temporary tables can significantly increase the compaction throughput. Avoid compaction of these tables.

Initiating automatic compaction in Cloudera Manager

Several properties in the Hive and Hive metastore service configurations must be set to enable automatic compaction. You need to check that the property settings are correct and to add one of the properties to the Hive on Tez service. Automatic compaction will then occur at regular intervals, but only if necessary.

About this task

Initiator threads should run in only one Hive Metastore server (even in high-availability / HA configurations). The Hive metastore instances will elect a single leader among themselves. There is no need to override the hive.compact or.initiator.on on the Hive metastore instance level. For more information, see *Hive Metastore leader election*.

Disable Initiator threads in all the Cloudera Data Hub clusters' Hive service, the compaction initiator thread can be run by the leader HMS in the DataLake cluster. Set the following properties:

- In Hive metastore (Hive-1) service:
  - hive.compactor.initiator.on = true (default)
- In Hive on Tez service:
  - hive.compactor.worker.threads = <a value greater than 0> (default and recommended value = 5)

Before you begin

Tables or partitions you are compacting must be full ACID or insert-only ACID tables.

Procedure

1. In Cloudera Manager, select the Hive metastore service: Clusters Hive-1 Configuration .
2. Search for compact.
3. Check that Turn on Compactor Initiator Thread (hive.compactor.initiator.on), Number of Threads Used by Compactor (hive.compactor.worker.threads).
4. Save the changes.
5. In Cloudera Manager, select the Hive on Tez service: Clusters HIVE_ON_TEZ-1 Configuration .
6. Search for compact.
7. Check that the Number of Threads Used by Compactor (hive.compactor.worker.threads).
8. Save the changes and restart the Hive on Tez and Hive (HIVE-1) metastore services at an appropriate time.

Related Information

Hive Metastore leader election

Starting compaction manually

You manually start compaction when automatic compaction fails for some reason. You can start compaction by running a Hive statement.

About this task

You can run compaction pseudo-synchronously using the AND WAIT clause. Compaction actually occurs asynchronously, but seems synchronous. The compaction request is recorded and queued, and remains in a waiting cycle, querying the status of the compaction in the background until a failure, success, or timeout occurs. The hive .compactor.wait.timeout (default: 300s) property sets the timeout.

Start compaction using a query

You use the following syntax to issue a query that starts compaction:

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

Before you begin

- Tables or partitions you are compacting must be full ACID or insert-only ACID tables.
- Compaction must be enabled (initiator hive.compactor.initiator.on=true)

Procedure

1. Run a query to start a major compaction of a table.

   ```
   ALTER TABLE mytable COMPACT 'major'
   ```

   Use the COMPACT 'minor' clause to run a minor compaction. ALTER TABLE compacts tables even if the NO_A UTO_COMPACTION table property is set.
2. Start compaction in a pseudo-synchronous way.

   ```
   ALTER TABLE mydb.mytable PARTITION (mypart='myval') COMPACT 'MAJOR' AND
   WAIT;
   ```

Options to monitor compactions

You can view the progress of compactions using the SHOW COMPACTIONS statement or by querying the COMP ACTIONS view within the SYS database.

The query statement displays the following details about compactions:

- C_ID: Unique internal compaction ID
- C_DATABASE: Hive database name
- C_TABLE: Table name
- C_PARTITION: Partition name (if the table is partitioned)
- C_TYPE: Major or minor compaction
- C_STATE: Compaction state
  - initiated: waiting in queue to be compacted
  - working: currently being compacted
  - ready for cleaning: compaction completed and old files scheduled for removal
  - failed: compaction job failed. Details are printed to the metastore log.
  - succeeded: compaction job completed successfully
  - attempted: initiator attempted to schedule a compaction but failed or hive.compactor.initiator.failed.compacts .threshold is breached and compaction is skipped . Details are printed to the metastore log.
- C_WORKER_HOST: Thread ID of the worker thread performing the compaction
- C_WORKER_ID: Unique internal worker ID
- C_WORKER_VERSION: Version of distribution with running worker
- C_ENQUEUE_TIME: Time when compaction request was placed
- C_START: Start time of the compaction job
- C_DURATION: Duration (in ms) of the compaction job
- C_HADOOP_JOB_ID: ID of the submitted MapReduce job for MR compaction. 'None' for query-based compaction.
- C_INITIATOR_HOST: Host where initiator is configured
- C_INITIATOR_ID: Unique internal initiator ID

- C_INITIATOR_VERSION: Version of distribution with active initiator

Note: Details about "initiators" and "workers" are new additions that you can use to investigate the compaction setup in an environment. If you are unable to retrieve these details using the options listed here, you can run the following query on the backend database:

```
SELECT activity.pid,
       activity.usename,
       activity.client_addr,
       activity.client_hostname,
       activity.query,
       blocking.pid AS blocking_pid,
       blocking.query AS blocking_query,
       blocking.client_addr AS blocking_client_addr,
       blocking.client_hostname AS blocking_client_hostname
FROM pg_stat_activity AS activity
JOIN pg_stat_activity AS blocking
    ON blocking.pid = ANY(pg_blocking_pids(activity.pid))
WHERE blocking.query like '%AUX_TABLE%';
```

The following sections describe the various options that you can use to monitor compactions.

SHOW COMPACTIONS

You can run the SHOW COMPACTIONS statement to view details about all the compaction jobs.

SHOW COMPACTIONS;

Authorization is not required to use the SHOW COMPACTIONS statement. Therefore, every user can view compactions and their current state.

Since the statement lists all the compaction jobs, you cannot filter or limit the results as required. Alternatively, you can use the SYS database to query and filter specific compactions.

Querying the SYS database

The SYS database in the Hive metastore contains the following sources that you can use to monitor compactions:

- COMPACTION_QUEUE: Stores information about compaction requests - both explicit user submitted and compactions discovered by the initiator and enqueued.
- COMPLETED_COMPACTIONS: Stores the same information present in COMPACTION_QUEUE along with the compaction's end time. Records are moved to the completed table when the cleaner finishes the job on the compaction request.
- COMPACTIONS: A view over the COMPACTION_QUEUE and COMPLETED_COMPACTIONS tables that resolves the column values to human readable format.

The SYS database tables and views are treated as normal Hive external tables or views, and therefore, standard access policies can be configured against these sources.

The following examples illustrate how you can run queries on the SYS.COMPACTIONS view to monitor compactions:

Query to display the last 10 failed compactions

```
SELECT *
FROM SYS.COMPACTIONS
WHERE C_STATE='failed'
ORDERBY C_ID DESC
LIMIT 10;
```

Query to check the status of a specific compaction using the compaction ID

```
SELECT *
```

```
 FROM SYS.COMPACTIONS
 WHERE C_ID='1234';
```

Query to view the total number of compactions in a particular state

```
SELECT COUNT(*)
 FROM SYS.COMPACTIONS
 WHERE C_STATE='ready for cleaning';
```

Note: Similarly, you can also query the INFORMATION_SCHEMA database for details about compactions. You must use the Ranger service and set up access policies for Hive users on this database to make it accessible.

Disabling automatic compaction

You can disable automatic compaction of a particular Hive ACID table by setting a Hive table property. By default, compaction is enabled, so you must enter an ALTER TABLE command to disable it.

About this task

Compaction of a full ACID table is skipped under the following conditions:

- Another compaction is either running or already initiated for the target.
- The compaction target table is already dropped.
- Table is explicitly configured to be skipped by the auto-compaction

Compaction of an insert-only, ACID table is skipped if hive.compactor.compact.insert.only is set to false (turned off). The default is true. Although you can disable automatic compaction, tables still can be compacted if you explicitly request compaction.Disabling automatic compaction does not prevent you from performing manual compaction.

The compaction auto-initiator can be disabled on service instance level (disabled by default). You can independently enable or disable compaction workers on service instance level. Compaction merges only the bucket files with the same index and keeps the same number of bucket files in base. Compaction does not rebalance the rows between the buckets.

Procedure

At the Hive JDBC client prompt, in the database of the target table, alter the TBLPROPERTIES.

```
ALTER TABLE my_t SET TBLPROPERTIES ('NO_AUTO_COMPACTION'='true');
```

Configuring compaction using table properties

You see how to configure compaction using table properties and learn about the advantage of using this method of configuration.

About this task

You can configure compaction using table properties. Using table properties, you can group all table and partition compactions into a specific queue to match your use case. You can also size the compactor job based on the tables parameters like size and compression.

Procedure

Set table properties to adjust the compaction initiator properties.

```
ALTER TABLE mydb.mytable
SET TBLPROPERTIES (
'compactorthreshold.hive.compactor.delta.pct.threshold'='0.2f',
'compactorthreshold.hive.compactor.delta.num.threshold'='20');
```

These properties change thresholds, as the names imply: the deltas/base size percentage override threshold and the number of deltas threshold.

Configuring compaction in Cloudera Manager

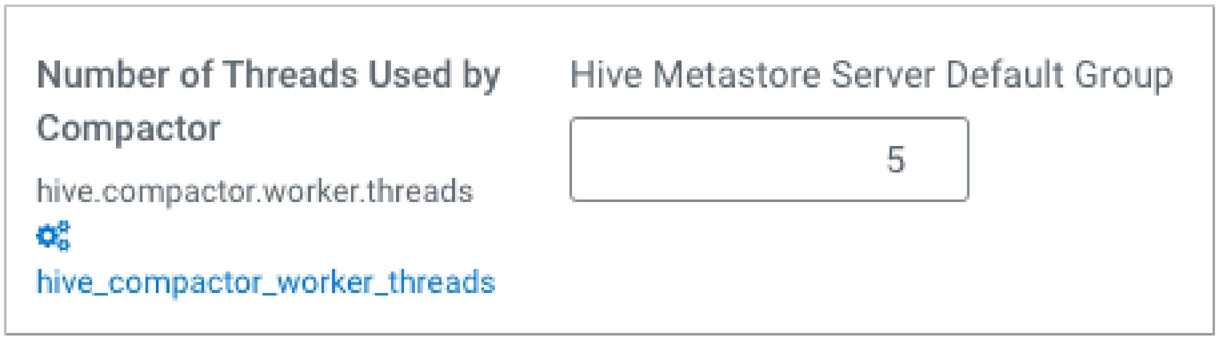
You see how to configure compaction properties exposed in Cloudera Manager. You rarely need to configure other compaction properties, but need to know how to do this.

About this task

You can configure Hive metastore (HMS) and HiveServer (HS2) properties. Compaction properties you typically configure appear in Cloudera Manager. You search for a property to change it. Occasionally, you might add a property to hive-site or core-site using the Cloudera Manager Safety Valve.

Procedure

1. In Cloudera Manager, click Clusters Hive Configuration , and search for a property, such as hive.compactor.worker.threads.



Number of Threads Used by Compactor
hive.compactor.worker.threads
hive_compactor_worker_threads
Hive Metastore Server Default Group
5

2. Change the value, and save.
3. Restart the Hive cluster.

Configuring the compaction check interval

You need to know when and how to control the compaction process checking, performed in the background, of the file system for changes that require compaction.

When you turn on the compaction initiator, consider setting the hive.compactor.check.interval property. This property determines how often the initiator should search for possible tables, partitions, or compaction. By default, the value for this property is set to 300 seconds. Decreasing hive.compactor.check.interval has the following effect:

- Reduces the time it takes for compaction to be started for a table or partition that requires compaction.
- Requires several calls to the file system for each table or partition that has undergone a transaction since the last major compaction, resulting in increases to the load on the filesystem.

The compaction initiator first checks the completed transactions (COMPLETED_TXN_COMPONENTS), excluding those that already have completed compactions, searching for potential compaction targets. The search of the first iteration includes all transactions. Further searching of iterations are limited to the time-frame since the last iteration.

To configure the compaction check interval, set the hive.compactor.check.interval. For example:

From Cloudera Manager, go to Clusters Hive Metastore Configuration and set the value for the hive.compactor.c heck.interval parameter in the Hive Metastore Server Advanced Configuration Snippet (Safety Valve) for hive-site.xml file.

Compactor properties

You check and change a number of Apache Hive properties to configure the compaction of delta files that accumulate during data ingestion. You need to know the defaults and valid values.

Basic compactor properties

hive.compactor.initiator.on

Default=false

Whether to run the initiator threads on this metastore instance or not. Set in only one instance.

hive.compactor.cleaner.on

Default=false

Whether to run the cleaner threads on this metastore instance or not. Set in only one instance.

Note: Before HIVE-26908, the cleaner thread could only be enabled or disabled through the hive.compactor.initiator.on property. With the introduction of the new cleaner property, you can enable or disable the cleaner and initiator threads independently.

hive.compactor.worker.threads

Default=0

Set this to a positive number to enable Hive transactions, which are required to trigger transactions. Worker threads spawn jobs to perform compactions, but do not perform the compactions themselves. Increasing the number of worker threads decreases the time that it takes tables or partitions to be compacted. However, increasing the number of worker threads also increases the background load on the Cloudera cluster because they cause more jobs to run in the background.

hive.metastore.runworker.in

Default=HS2

Specifies where to run the Worker threads that spawn jobs to perform compactions. Valid values are HiveServer (HS2) or Hive metastore (HMS).

hive.compactor.abortedtxn.threshold

Default=1000 aborts

The number of aborted transactions that triggers compaction on a table/partition.

hive.compactor.aborted.txn.time.threshold

Default=12 hours

The hours of aborted transactions that trigger compaction on a table/partition.

Advanced compactor properties

hive.compactor.worker.timeout

Default=86400s

Expects a time value using d/day, h/hour, m/min, s/sec, ms/msec, us/usec, or ns/nsec (default is sec). Time in seconds after which a compaction job will be declared failed and the compaction re-queued.

hive.compactor.check.interval

Default=300s

A valid value is a time with unit (d/day, h/hour, m/min, s/sec, ms/msec, us/usec, ns/nsec), which is sec if not specified.

Time in seconds between checks to see if any tables or partitions need to be compacted. This value should be kept high because each check for compaction requires many calls against the NameNode. Decreasing this value reduces the time it takes to start compaction for a table or partition that requires it. However, checking if compaction is needed requires several calls to the NameNode for each table or partition involved in a transaction done since the last major compaction. Consequently, decreasing this value increases the load on the NameNode.

hive.compactor.delta.num.threshold

Default=10

Number of delta directories in a table or partition that triggers a minor compaction.

hive.compactor.delta.pct.threshold

Default=0.1

Percentage (fractional) size of the delta files relative to the base that triggers a major compaction. (1.0 = 100%, so the default 0.1 = 10%.)

hive.compactor.max.num.delta

Default=500

Maximum number of delta files that the compactor attempts to handle in a single job.

hive.compactor.wait.timeout

Default=300000

The value must be greater than 2000 milliseconds.

Time out in milliseconds for blocking compaction.

hive.compactor.initiator.failed.compacts.threshold

Default=2

A valid value is between 1 and 20, and must be less than hive.compactor.history.retention.failed.

The number of consecutive compaction failures (per table/partition) after which automatic compactions are not scheduled any longer.

hive.compactor.cleaner.run.interval

Default=5000ms

A valid value is a time with unit (d/day, h/hour, m/min, s/sec, ms/msec, us/usec, ns/nsec), which is msec if not specified.

The time between runs of the cleaner thread.

hive.compactor.job.queue

Specifies the Hadoop queue name to which compaction jobs are submitted. If the value is an empty string, Hadoop chooses the default queue to submit compaction jobs.

Providing an invalid queue name results in compaction job failures.

hive.compactor.compact.insert.only

Default=true

The compactor compacts insert-only tables, or not (false). A safety switch.

hive.compactor.crud.query.based

Default=false

Performs major compaction on full CRUD tables as a query, and disables minor compaction.

hive.split.grouping.mode

Default=query

A valid value is either query or compactor.

This property is set to compactor from within the query-based compactor. This setting enables the Tez SplitGrouper to group splits based on their bucket number, so that all rows from different bucket files for the same bucket number can end up in the same bucket file after the compaction.

hive.compactor.history.retention.succeeded

Default=3

A valid value is between 0 and 100.

Determines how many successful compaction records are retained in compaction history for a given table/partition.

hive.compactor.history.retention.failed

Default=3

A valid value is between 0 and 100.

Determines how many failed compaction records are retained in compaction history for a given table/partition.

hive.compactor.history.retention.attempted

Default=2

A valid value is between 0 and 100.

Determines how many attempted compaction records are retained in compaction history for a given table/partition.

hive.compactor.history.reaper.interval

Default=2m

A valid value is a time with unit (d/day, h/hour, m/min, s/sec, ms/msec, us/usec, ns/nsec), which is msec if not specified.

Determines how often the compaction history reaper runs.

Compaction observability in Cloudera Manager

Compaction observability is a notification and information system based on metrics about the health of the compaction process. A healthy compaction process is critical to query performance, availability, and uptime of your data warehouse. You learn how to use compaction observability to prevent serious problems from developing.

Compaction runs in the background. At regular intervals, Hive accesses the health of the compaction process, and logs an error in the event of a problem. The assessment is based on metrics, such as the number of rows in the metadata table TXN_TO_WRITE_ID and the age of the longest running transaction (oldest_open_txn_age_in_sec). For example, if compaction is not running, the TXN_TO_WRITE_ID table in the HMS backend database becomes bloated and queries slow down.

Compaction health monitoring provides the following information to help you proactively address the problems before the problems become an emergency:

- Warnings and errors indicating problems in compaction processes
- Charts of various metrics that provide information about compaction health
- Recommended actions to address suboptimal configurations

Compaction observability does not attempt to do root cause analysis (RCA) and does not attempt to fix the underlying problem. Compaction observability helps you quickly respond to symptoms of compaction problems.

Using Cloudera Manager, you can view compaction health checks for the Hive Metastore and Hive on Tez services, view actions and advice related to configurations and thresholds, and use the Compaction tab from the Hive Metastore service to view compaction-related charts based on the collected metrics.

Configuring compaction health monitoring

As an administrator, you can use Cloudera Manager to enable or disable compaction health tests for the Hive Metastore (HMS) and HiveServer (HS2) services, configure how those health tests factor into the overall health of the service, and modify thresholds for the status of certain health tests.

About this task

You can configure HMS and HS2 compaction properties that are exposed in Cloudera Manager by searching for the property. Occasionally, you might add a property to hive-site.xml or core-site.xml using the Hive Service Advanced Configuration Snippet (Safety Valve).

Procedure

1. In Cloudera Manager, click Clusters Hive Configuration to navigate to the configuration page for HMS.
2. Search for Hive Compaction Health Test, enable the property, and save the changes.
3. Enable the Hive Compaction Health Test property for the HS2 service by going to Clusters Hive on Tez Configuration .
4. To configure the Health Check thresholds, go to Clusters Hive Configuration .
5. Select the Category Monitoring filter.
6. Modify the warning and critical (error) threshold values for the required compaction-related thresholds and save the changes.

Results

The **Health Tests** panel of HMS and HS2 service displays the compaction health test results, typically with specific metrics that the test returned.

Note: For the HS2 service, if there is no delta metric data (metric values are too low to be collected), then the health check displays a "Disabled" status.

Monitoring compaction health in Cloudera Manager

Using Cloudera Manager, you can view compaction health checks for the Hive Metastore (HMS) and Hive on Tez services, view actions and advice related to configurations and thresholds, and use the Compaction tab from the Hive Metastore service to view compaction-related charts based on the collected metrics.

Before you begin

- You must have enabled compaction health checks.
- You must have enabled the Hive ACID metrics service to collect ACID-related metrics.

Procedure

1. Sign in to Cloudera Manager as an Administrator.
2. Go to the HMS service by clicking Clusters Hive .
   The **Health Tests** panel in the **Status** tab displays the compaction health test results, typically with the specific metrics that the test returned.

   Similarly, you can view the compaction health test results for the HiveServer (HS2) service by going to Clusters Hive on Tez .
3. From the HMS service page, click the Compactions tab to view dashboards of compaction-related charts of various metrics.

   You can create triggers for the charts that allow you to define actions to be taken when a specified condition is met.

Hive ACID metric properties for compaction observability

As an administrator, you must configure certain properties related to the Hive ACID metrics service to enable the collection of Hive ACID metrics that are required to display compaction-related alerts and charts in Cloudera Manager.

Basic properties

You must configure the following metrics to enable Hive ACID metrics collection:

hive.metastore.metrics.enabled

Default value: True

Enables or disables Hive metrics subsystem for the Hive Metastore (HMS) role.

hive.server2.metrics.enabled

Default value: True

Enables or disables Hive metrics subsystem for the HiveServer (HS2) role.

hive.metastore.acidmetrics.ext.on

Default value: True

Set this to property to True to collect additional acid related metrics outside of the Hive ACID metrics service. Enable this property for both HMS and HS2 roles.

Note: Ensure that you have also enabled the hive.metastore.metrics.enabled property for HMS and the hive.server2.metrics.enabled property for HS2.

metastore.acidmetrics.thread.on

Default value: True

Specifies whether to run Hive ACID related metrics collection on this metastore instance. Enable this property in only one HMS instance and disable the property in all other instances.

Advanced properties

Set these additional properties to configure how Hive ACID metrics service collects metric data:

The following properties have to be configured in the HMS instance where metastore.acidmetrics.thread.on is enabled:

metastore.acidmetrics.check.interval

Default value: 300 seconds

Specifies the time (in seconds) between Hive ACID metric collection jobs.

metastore.acidmetrics.table.aborted.txns.threshold

Default value: 1500

Hive metrics subsystem collects the number of tables that have a large number of aborted transactions. This property specifies the minimum number of aborted transactions required to consider a table.

metastore.compactor.acid.metrics.logger.frequency

Default value: 360m

Specifies the logging frequency (in minutes) of Hive ACID metrics. Set this property in only one HMS instance and set the property to 0 in all other HMS instances to disable logging.

The following properties have to be configured in the HS2 (Hive on Tez) service:

hive.txn.acid.metrics.cache.size

Default value: 100

Specifies the size of the ACID metrics cache, which is the maximum number of partitioned and unpartitioned tables with the most deltas that is included in the list of active, obsolete, and small deltas. Valid values are between 0 to 500.

hive.txn.acid.metrics.cache.duration

Default value: 7200 seconds

Specifies the maximum lifetime (in seconds) for an entry in the ACID metrics cache.

hive.txn.acid.metrics.reporting.interval

Default value: 30 seconds

Specifies the reporting period (in seconds) for ACID metrics.

hive.txn.acid.metrics.delta.num.threshold

Default value: 100

Specifies the minimum number of active delta files that a table or partition must have to be included in the ACID metrics report.

hive.txn.acid.metrics.obsolete.delta.num.threshold

Default value: 100

Specifies the minimum number of obsolete delta files that a table or partition must have to be included in the ACID metrics report.

hive.txn.acid.metrics.delta.check.threshold

Default value: 300 seconds

Specifies the minimum age (in seconds) for delta files to be included in the ACID metrics report. Delta files less than this age are not included in the report.

hive.txn.acid.metrics.delta.pct.threshold

Default value: 0.01

Specifies the percentage size of the delta files relative to the base directory. Delta files whose size is smaller than this threshold value are considered as small deltas and are not included in the ACID metrics report. (1 = 100%, so the default 0.01 = 1%)

Query vectorization

You can use vectorization to improve instruction pipelines for certain data and queries and to optimize how Hive uses the cache. Vectorization processes batches of primitive types on the entire column rather than one row at a time.

Unsupported functionality on vectorized data

Some functionality is not supported on vectorized data:

- DDL queries
- DML queries other than single table, read-only queries
- Formats other than Optimized Row Columnar (ORC)

Supported functionality on vectorized data

The following functionality is supported on vectorized data:

- Single table, read-only queries

  Selecting, filtering, and grouping data is supported.
- Partitioned tables

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

Supported data types

You can query data of the following types using vectorized queries:

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

Vectorization default

Vectorized query execution can affect performance. You need to be aware of the Boolean default value of hive.vectorized.execution.enabled.

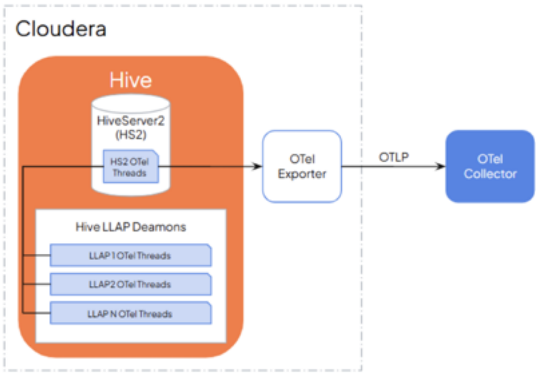
Vectorized query execution is enabled by default (true). Vectorized query execution processes Hive data in batch, channeling a large number of rows of data into columns, foregoing intermediate results. This technique is more efficient than the MapReduce execution process that stores temporary files.

OpenTelemetry support for Hive

Learn how Hive uses OpenTelemetry (OTel) to collect and publish telemetry data, processed by a user-configured OTel collector for visualization in systems like Jaeger and Prometheus.

Overview

As part of this offering, Hive in Cloudera Runtime7.3.1.500 SP3 includes an OTel exporter that helps to collect, filter, and publish telemetry information, such as infrastructure and workload metrics, live and historical query data.



HiveServer2 and LLAP each transmit telemetry data to an OTel agent. The OTel agent independently transmits the data to a customer configured OTel collector instance for processing. The processed data can be exported and visualized through backend systems, such as Jaeger, Zipkin, Prometheus.

Note: The OTel collector is not part of the Cloudera Runtime 7.3.1.500 SP3 Cloudera Data Warehouse deployment. You must have your own instance of the OTel collector that is configured with backend instances (Jaeger, Zipkin, Prometheus).

HiveServer2 and LLAP integration with OTel

OTel threads function independently in both HiveServer2 and LLAP daemons. When query execution begins, these threads capture essential telemetry data, which is transmitted to an OTel collector based on a configurable recurring schedule. The collected data is then processed and forwarded to backend systems for visualization.

HiveServer2 integrates with the OTel agent to expose both query-related data and JVM metrics. A dedicated thread or service runs within HiveServer2 to handle this integration. This thread collects query details and metrics, which are then transmitted through the OTel agent. These transmitted metrics can be collected by OTel collectors for analysis and visualization.

Metrics specific to each LLAP daemon, such as JVM and memory-related statistics, are also transmitted for detailed observability.

Related Information

OpenTelemetry

Benefits of OTel Hive Integration

This topic explains the benefits of integrating OpenTelemetry (OTel) with HiveServer2 and LLAP, focusing on enhanced observability, telemetry data insights, and optimized performance.

The integration of OpenTelemetry (OTel) with HiveServer2 and LLAP provides advanced telemetry capabilities, enabling better observability and diagnostics while maintaining optimal system performance.

Telemetry Data Exposed

OTel integration allows HiveServer2 and LLAP to expose the following telemetry data through an OTel collector:

- Metrics: Infrastructure and workload metrics, such as JVM memory usage, thread counts, and Operating System-related insights.
- Live Query Data: Tracking of active query lifecycle events, including execution times, stages, and error messages.
- Historical Query Data: Detailed query execution metadata for analysis and diagnostics.

Performance Benefits

The OTel integration is designed to enhance observability while ensuring HiveServer2 and LLAP maintain optimal performance.

Optimized for Minimal Impact: Integrating OTel with HiveServer2 and LLAP ensures seamless performance. By utilizing an independent thread or service to collect, translate, and expose metrics already tracked by HiveServer2, the integration ensures query execution remains smooth, without any noticeable delays or disruptions.

Scalable for Future Metrics: The system is well-prepared to handle new metrics if introduced. While tracking additional metrics may require some extra resources, this scalable design ensures the system remains efficient, with any potential impact being manageable and dependent on your specific use case.

Efficient Memory Usage: The OTel thread is optimized for minimal memory consumption, efficiently managing the receive, translate, and expose phases. The memory usage is small and well within acceptable limits, ensuring it doesn't affect overall system performance, even during extended operations.

Configuring OTel in HiveServer2

Learn how you can enable OTel in HiveServer2 and configure certain properties that will enable you to optimize the data collection.

Before you begin

Ensure that you are on Cloudera Runtime 7.3.1.500 or higher version.

Procedure

1. In Cloudera Manager, click Clusters Hive on Tez Configuration .
2. Search for **HiveServer2 Advanced Configuration Snippet (Safety Valve) for hive-site.xml**
3. Modify the values as required:

   a) hive.otel.metrics.frequency.seconds (Default: 0s)

      Specifies the frequency at which telemetry data is transmitted to the OTel collector. By default, the value is set to 0 seconds indicating that OpenTelemetry data collection is disabled. Enter a value greater than 0 to enable OpenTelemetry.

      If the value is 5s. This indicates that telemetry data is transmitted to the OTel collector every 5 seconds.

   b) hive.otel.collector.endpoint

      Specifies the endpoint where all the OpenTelemetry Protocol (OTLP) traces and metrics are transmitted. The endpoint represents the address of an OTel collector. The endpoint must be a valid URL with https scheme.

      https://<otel-collector-host>:<port>

      Important: Cloudera recommends using https to ensure secure data transmission.

   c) hive.otel.exporter.timeout (Default: 10m)

      Specifies the maximum time allowed for the OTel agent to complete a transmit operation. The transmit operation times out if it exceeds the specified time.

   d) hive.otel.retry.initial.backoff (Default: 10s)

      Specifies the initial time delay before attempting to retry a failed transmit operation. The value serves as the starting point for the exponential backoff strategy.

   e) hive.otel.retry.max.backoff (Default: 1m)

      Specifies the maximum time that the OTel agent should wait between retries. This sets an upper limit on the backoff interval ensuring that retry export operations do not exceed the specified duration even with exponential backoff.

   f) hive.otel.retry.backoff.multiplier (Default: 5f)

      Specifies the factor by which the retry interval increases after every failed attempt. This determines how much the backoff interval increases after each failed attempt, following an exponential backoff strategy.
4. Save the changes and restart the Hive service

Telemetry data exposed to OTel collector for hive

HiveServer2 transmits telemetry data related to live queries, completed queries, task-level details, JVM metrics related to memory usage and thread count, and Operating System level statistics.

Query and Task related insights

Live queries

The following metrics related to live HiveServer2 queries are transmitted to the OTel collector:

Metrics	Description
QueryId	Represents the unique identifier for the query.
QueryString	The SQL statement of the query.
UserName	The user who submitted the query.
ExecutionEngine	The query execution engine used to process the query, typically Tez.
ErrorMessage	Errors encountered during query execution.

Completed queries

The following metrics related to completed HiveServer2 queries are transmitted to the OTel collector:

Metrics	Description
QueryId	Represents the unique identifier for the query.
QueryStartTime	Timestamp when the query execution started.
EndTime	Timestamp when the query execution completed.
OperationId	Unique identifier for tasks related to query execution. For example, a0fe8acc-6b9a-4f54-8537-f7b3bf7dea72
OperationLogLocation	Path to the local log file for additional query log entries.
ErrorMessage	Errors encountered during query execution.
ExplainPlan	Output showing how the engine executed the query.
FullLogLocation	Location of the complete application logs related to the query.
Running	Indicates whether the query is currently in progress.
Runtime	Duration of the query execution in seconds or minutes.
UserName	The user who submitted the query.
ExecutionEngine	The query execution engine used to process the query, typically Tez.
State	Current state of the query (e.g., RUNNING, SUCCESS, ABORTED, or FAILED).
SessionId	Identifier for the session in which the query was executed.

Task-level details

The following metrics related to tasks are transmitted to the OTel collector:

Metrics	Description
TaskId	Unique identifier for each task.

Metrics	Description
Name	Task types such as MAPRED (Mapper/Reducer tasks), DEPENDENCY_COLLECTION, or STATS TASK. If multiple tasks exist, integers are appended.
TaskType	Representation with unique values while execution such as MAPRED, DEPENDENCY_COLLECTION, MOVE, or STATS.
Status	Current task status (e.g. Success).
StatusMessage	Status with detailed task information
ExternalHandle	DAGID, used for query optimization and execution.
ErrorMsg	Error details if the task failed.
ReturnValue	Task result. A value of 0 indicates success, while negative integers indicate an issue.
BeginTime	Time when the task execution started.
ElapsedTime	Time spent executing the task.
EndTime	Time when the task execution finished.

Java Virtual Machine (JVM) metrics

HiveServer2 collects various JVM metrics, including:

- Memory usage: Data such as heap and non-heap memory usage.
- Thread count: Counts of threads in different states (for example, runnable, waiting).
- OS-Level statistics: CPU load, memory size, and swap space details.

Metrics related to memory usage

The following metrics related to JVM heap and non-heap memory usage are transmitted to the OTEL collector:

Metrics	Description
memNonHeapUsedMGauge	Size (in MB) of non-heap memory currently used by the JVM.
memNonHeapCommittedM	Amount of non-heap memory (in MB) reserved by the JVM for internal use.
memNonHeapMaxM	Maximum allowable size (in MB) for non-heap memory.
memHeapUsedM	Size (in MB) of heap memory currently used by the JVM.
memHeapCommittedM	Size (in MB) of heap memory reserved by the JVM.
memHeapMaxM	Maximum allowable size (in MB) for heap memory.
memHeapMaxM	Maximum memory available (in MB) to the JVM.

Metrics related to thread count

The following metrics related to the JVM threads in different states (runnable, waiting) are transmitted to the OTel collector:

Metrics	Description
threadsNew	Number of threads currently in the "NEW" state within a Java application.
threadsRunnable	Number of threads ready to run or currently executing on the CPU.

Metrics	Description
threadsBlocked	Number of threads waiting to acquire a lock.
threadsWaiting	Number of threads waiting indefinitely for a signal from another thread.
threadsTimedWaiting	Number of threads waiting for a specific duration before proceeding.
threadsTerminated	Number of threads where execution is completed.

OS-level statistics

The following OS related metrics, such as CPU load, memory size, swap space details are transmitted to the OTel collector:

Note: These metrics are transmitted only for Unix-based operating systems.

Metrics	Description
systemCpuLoad	Measures the CPU load of the host machine, container, or pod.
committedVirtualMemorySize	Amount of allocated memory, including both physical RAM and virtual memory reserved by the operating system.
processCpuTime	Total CPU time consumed by a specific process or thread.
freePhysicalMemorySize	Amount of unused physical memory (RAM) available to processes and the operating system, container, or pod.
freeSwapSpaceSize	Amount of swap space currently available.
totalPhysicalMemorySize	Total installed physical memory (RAM) in the system.
processCpuLoad	Percentage of CPU load consumed by a specific process.

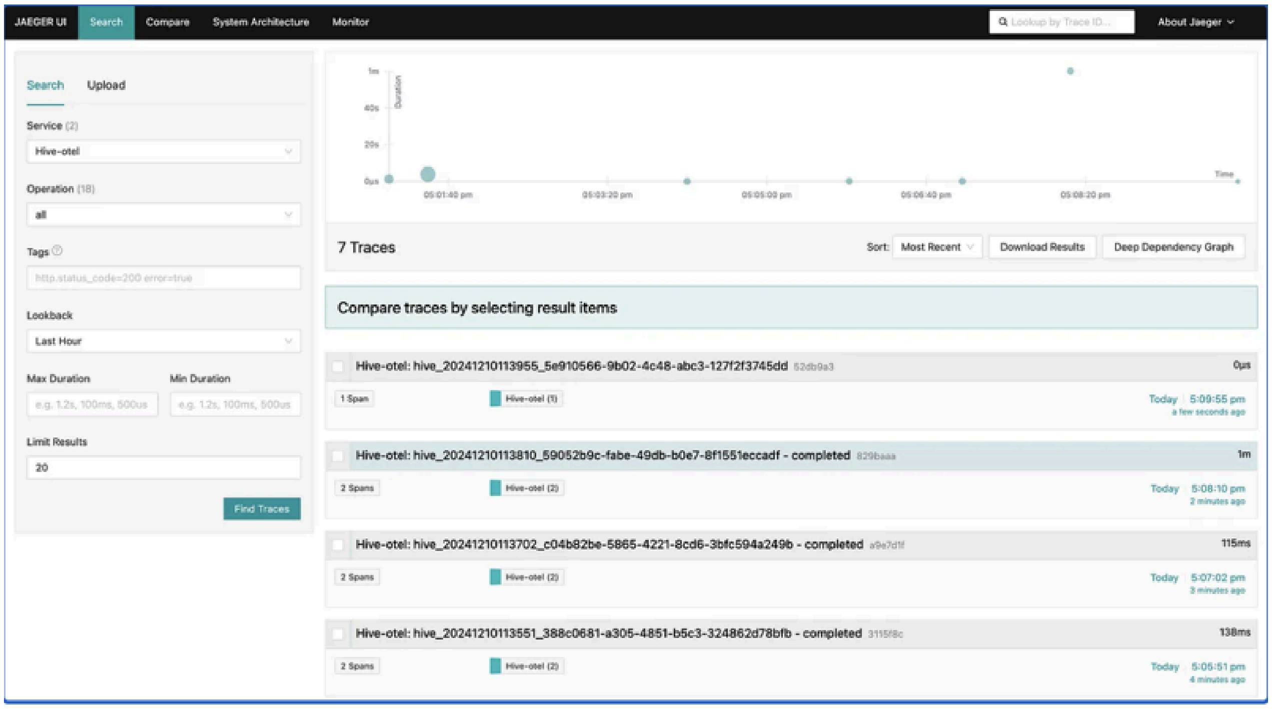
Example Visualizations through OTel backend

This section provides some examples of OTel backend systems that allow you to view the telemetry data that is processed and exported from an OTel collector.

Important: The example backend systems shown here are just for representation. Along with having your own instance of the OTel collector, you must also have your own instances of Jaeger, Zipkin, Prometheus and Grafana to view the telemetry data.

Visualizing through Jaeger

The following image represents a Jaeger UI displaying traces for 4 queries out of which 1 is a live query and the remaining 3 are completed queries. The completed queries are denoted by the completed suffix and the running queries are denoted by just the Query ID and do not have a suffix.



Service (2): Hive-otel
Operation (18): all
Tags: http.status_code=200 error=true
Lookback: Last Hour
Max Duration: e.g. 1.2s, 100ms, 500us
Min Duration: e.g. 1.2s, 100ms, 500us
Limit Results: 20

7 Traces
Sort: Most Recent

Compare traces by selecting result items

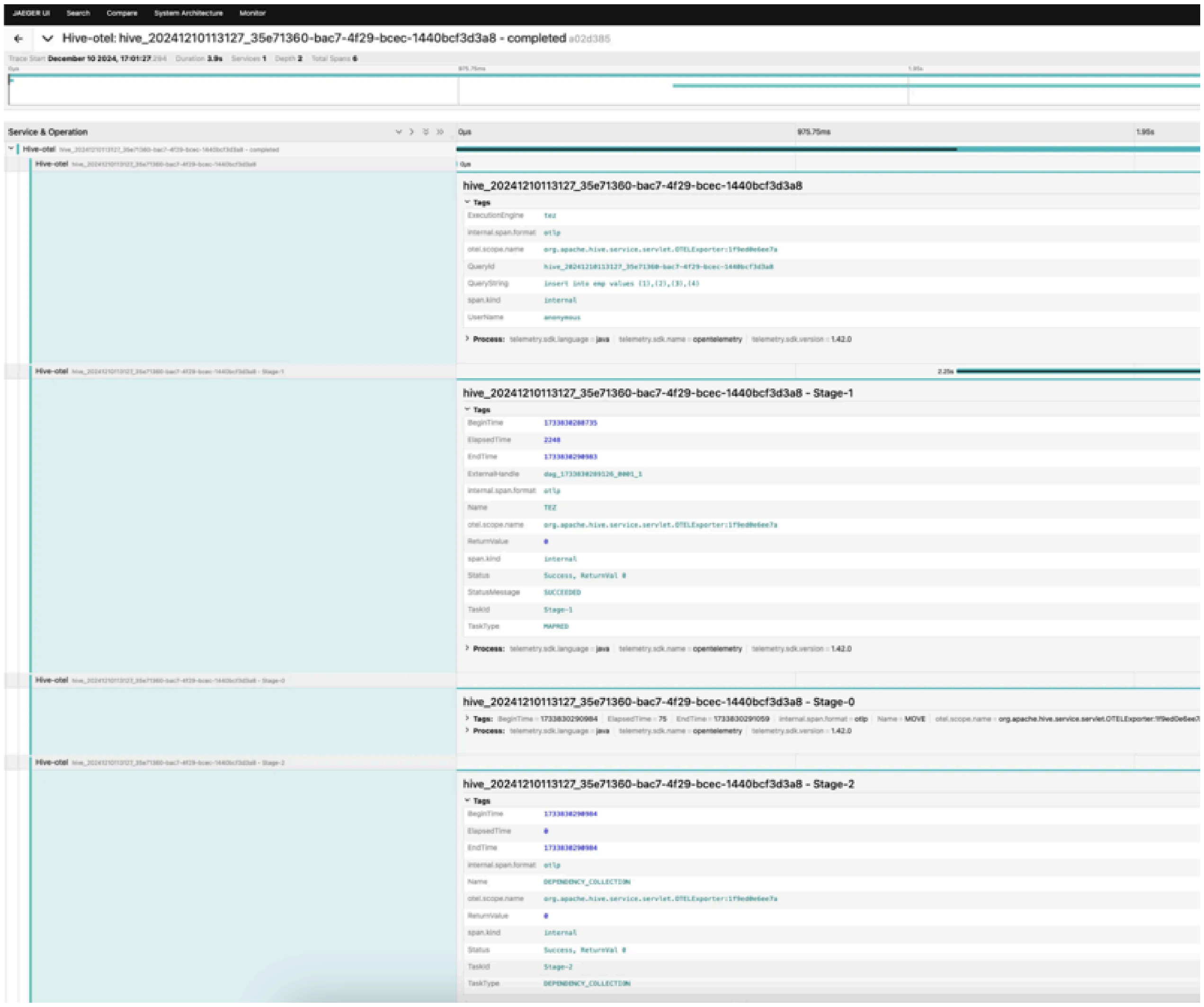
Hive-otel: hive_20241210113955_5e910566-9b02-4c48-abc3-127f2f3745dd 52db9a3 — 0µs
1 Span — Hive-otel (1) — Today 5:09:55 pm, a few seconds ago

Hive-otel: hive_20241210113810_59052b9c-fabe-49db-b0e7-8f1551eccadf - completed 829baaa — 1m
2 Spans — Hive-otel (2) — Today 5:08:10 pm, 2 minutes ago

Hive-otel: hive_20241210113702_c04b82be-5865-4221-8cd6-3bfc594a249b - completed a9e7d1f — 115ms
2 Spans — Hive-otel (2) — Today 5:07:02 pm, 3 minutes ago

Hive-otel: hive_20241210113551_388c0681-a305-4851-b5c3-324862d78bfb - completed 3115f8c — 138ms
2 Spans — Hive-otel (2) — Today 5:05:51 pm, 4 minutes ago

You can click on a trace to view more details about the attributes and tasks. This helps you understand task breakdowns and performance aspects of individual query components.



Hive-otel: hive_20241210113127_35e71360-bac7-4f29-bcec-1440bcf3d3a8 - completed a02d385

Trace Start December 10 2024, 17:01:27.294 | Duration 3.9s | Services 1 | Depth 2 | Total Spans 6

hive_20241210113127_35e71360-bac7-4f29-bcec-1440bcf3d3a8

Tags

ExecutionEngine	tez
internal.span.format	otlp
otel.scope.name	org.apache.hive.service.servlet.OTELExporter:1f9ed0e6ee7a
QueryId	hive_20241210113127_35e71360-bac7-4f29-bcec-1440bcf3d3a8
QueryString	insert into emp values (1),(2),(3),(4)
span.kind	internal
UserName	anonymous

Process: telemetry.sdk.language = java | telemetry.sdk.name = opentelemetry | telemetry.sdk.version = 1.42.0

hive_20241210113127_35e71360-bac7-4f29-bcec-1440bcf3d3a8 - Stage-1

Tags

BeginTime	1733830288735
ElapsedTime	2248
EndTime	1733830290983
ExternalHandle	dag_1733830289126_0001_1
internal.span.format	otlp
Name	TEZ
otel.scope.name	org.apache.hive.service.servlet.OTELExporter:1f9ed0e6ee7a
ReturnValue	0
span.kind	internal
Status	Success, ReturnVal 0
StatusMessage	SUCCEEDED
TaskId	Stage-1
TaskType	MAPRED

Process: telemetry.sdk.language = java | telemetry.sdk.name = opentelemetry | telemetry.sdk.version = 1.42.0

hive_20241210113127_35e71360-bac7-4f29-bcec-1440bcf3d3a8 - Stage-0

Tags: BeginTime = 1733830290984 | ElapsedTime = 75 | EndTime = 1733830291059 | internal.span.format = otlp | Name = MOVE | otel.scope.name = org.apache.hive.service.servlet.OTELExporter:1f9ed0e6ee7

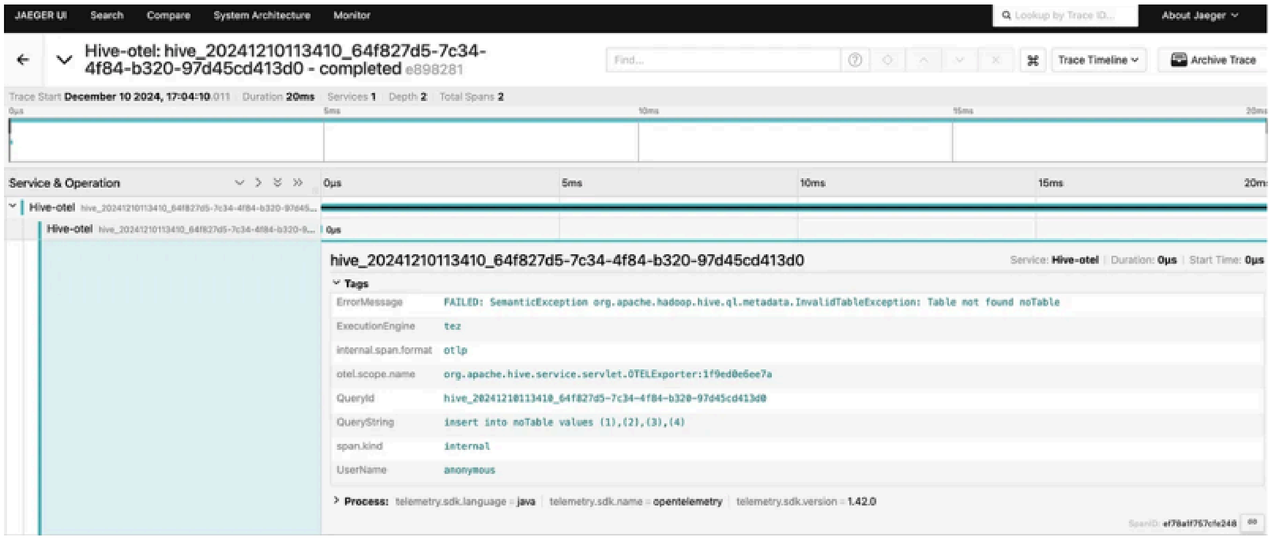
Process: telemetry.sdk.language = java | telemetry.sdk.name = opentelemetry | telemetry.sdk.version = 1.42.0

hive_20241210113127_35e71360-bac7-4f29-bcec-1440bcf3d3a8 - Stage-2

Tags

BeginTime	1733830290984
ElapsedTime	0
EndTime	1733830290984
internal.span.format	otlp
Name	DEPENDENCY_COLLECTION
otel.scope.name	org.apache.hive.service.servlet.OTELExporter:1f9ed0e6ee7a
ReturnValue	0
span.kind	internal
Status	Success, ReturnVal 0
TaskId	Stage-2
TaskType	DEPENDENCY_COLLECTION

Failed queries display an ErrorMessage in the expanded view enabling you to troubleshoot and debug effectively.



Hive-otel: hive_20241210113410_64f827d5-7c34-4f84-b320-97d45cd413d0 - completed e898281

Trace Start December 10 2024, 17:04:10.011 | Duration 20ms | Services 1 | Depth 2 | Total Spans 2

hive_20241210113410_64f827d5-7c34-4f84-b320-97d45cd413d0

Service: Hive-otel | Duration: 0μs | Start Time: 0μs

Tags

ErrorMessage	FAILED: SemanticException org.apache.hadoop.hive.ql.metadata.InvalidTableException: Table not found noTable
ExecutionEngine	tez
internal.span.format	otlp
otel.scope.name	org.apache.hive.service.servlet.OTELExporter:1f9ed0e6ee7a
QueryId	hive_20241210113410_64f827d5-7c34-4f84-b320-97d45cd413d0
QueryString	insert into noTable values (1),(2),(3),(4)
span.kind	internal
UserName	anonymous

Process: telemetry.sdk.language = java | telemetry.sdk.name = opentelemetry | telemetry.sdk.version = 1.42.0

SpanID: ef78a1f757cfe248

Limitation of OpenTelemetry support for Hive

Learn about the current limitations of OTel integration, including its focus on metrics and events, while future-proofing for logs. Understand the fixed nature of event data and its implications for query observability.

Telemetry Data Scope

The scope of telemetry data is currently limited to only metrics and events. Logs and traces are under consideration for a future release.

Fixed Event Data

The events being sent through an OTel agent are not configurable and can include Personally Identifiable Information (PII) within the SQL statement of the query.