

Managing Apache Hive

Date published: 2020-10-07

Date modified:



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

ACID operations.....	4
Configure partitions for transactions.....	4
View transactions.....	4
View transaction locks.....	5
Data compaction.....	6
Compaction prerequisites.....	7
Enable automatic compaction.....	7
Start compaction manually.....	8
View compaction progress.....	9
Disable automatic compaction.....	10
Compactor properties.....	10
Query vectorization.....	12
Tracking Hive on Tez query execution.....	13
Tracking an Apache Hive query in YARN.....	15
Application not running message.....	16

ACID operations

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. The Data Warehouse service in CDP Public Cloud includes mature versions of Hive ACID transaction processing and low latency analytical processing (LLAP). LLAP is not available in CDP Private Cloud Base. 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.

Configure 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.exec.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](#)

View transactions

As Administrator, you can view a list of open and aborted transactions.

Procedure

Enter a query to view transactions.

SHOW TRANSACTIONS

The following information appears in the output:

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

View transaction locks

As a Hive administrator, you can get troubleshooting information about locks on a table, partition, or schema.

About this task

Hive transactions, enabled by default, disables 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.

Before you begin

Check that transactions are enabled (the default).

Procedure

1. Enter a Hive query to check table locks.

```
SHOW LOCKS mytable EXTENDED;
```

2. Check partition locks.

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

3. Check schema locks.

```
SHOW LOCKS SCHEMA mydatabase;
```

The following information appears in the output unless ZooKeeper or in-memory lock managers are used.

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

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 housekeeping of files.

Hive creates a set of delta files for each transaction that alters a table or partition and stores them in a separate delta directory. By default, Hive automatically compacts delta and base files at regular intervals. Compaction is a consolidation of files. You can configure automatic compactions, as well as perform manual compactions of base and delta files. To submit compaction Jobs, Hive uses Tez as the execution engine, and uses MapReduce algorithms in the Stack. Compactions occur in the background without affecting concurrent reads and writes.

The compactor initiator should run on only one HMS instance.

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.

Related Information

[Apache Wiki transactions and compaction documentation](#)

Compaction prerequisites

To prevent data loss or an unsuccessful compaction, you must meet the prerequisites before compaction occurs.

Exclude compaction users from Ranger policies

Compaction causes data loss if Apache Ranger policies for masking or row filtering are enabled and the user hive or any other compaction user is included in the Ranger policies.

1. Set up Ranger masking or row filtering policies to exclude the user hive from the policies.

The user (named hive) appears in the Users list in the Ranger Admin UI.

Policy ID	Policy Name	Policy Labels	Status	Audit Logging	Roles	Groups	Users
7	all - hiveservice	--	Enabled	Enabled	--	--	hive rangerlookup impala

2. Identify any other compaction users from the masking or row filtering policies for tables as follows:
 - If the `hive.compaction.run.as.user` property is configured, the user runs compaction.
 - If a user is configured as owner of the directory on which the compaction will run, the user runs compaction.
 - If a user is configured as the table owner, the user runs compaction
3. Exclude compaction users from the masking or row filtering policies for tables.

Failure to perform these critical steps can cause data loss. For example, if a compaction user is included in an enabled Ranger masking policy, the user sees only the masked data, just like other users who are subject to the Ranger masking policy. The unmasked data is overwritten during compaction, which leads to data loss of unmasked content as only the underlying tables will contain masked data. Similarly, if Ranger row filtering is enabled, you do not see, or have access to, the filtered rows, and data is lost after compaction from the underlying tables.

The worker process executes queries to perform compaction, making Hive data subject to data loss ([HIVE-27643](#)). MapReduce-based compactons are not subject to the data loss described above as these compactons directly use the MapReduce framework.

Related Information

[Row-level filtering and column masking in Hive with Ranger policies](#)

Enable automatic compaction

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) in a public cloud environment. Disable Initiator threads in the other Hive Metastore servers in the DataLake cluster, and in all the Datahub clusters' Hive service, the compaction initiator thread can be run by a single Hive metastore in the DataLake cluster. The following properties must be set in Hive metastore (Hive-1) and Hive on Tez services as follows:

- `hive.compactor.initiator.on` = true (default)
- `hive.compactor.worker.threads` = <a value greater than 0> (default and recommended value = 5)
- `hive.metastore.runworker.in` = hs2 (default)

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.

HIVE-1 Actions

Apr 3, 10:57 PM UTC

Status Instances **Configuration** Commands Charts Library Audits Quick Links

Q compact Filters Role Groups History and Rollback

Filters

SCOPE

HIVE-1 (Service-Wide)	0
Gateway	0
Hive Metastore Server	3
HiveServer2	0
WebHCat Server	0

CATEGORY

Advanced	3
Cloudera Navigator	0

Turn on compactor initiator thread. ☒ Hive Metastore Server Default Group

hive.compactor.initiator.on

Number of Threads Used by Compactor Hive Metastore Server Default Group

hive.compactor.worker.threads

Run compactor on Hive Metastore or HiveServer2. Hive Metastore Server Default Group

hive.metastore.runworker.in ☐ metastore ☒ hs2

3. Check that Turn on Compactor Initiator Thread (hive.compactor.initiator.on), Number of Threads Used by Compactor (hive.compactor.worker.threads), and Run Compactor on Hive Metastore or HiveServer2 (hive.metastore.runworker.in) are set to the values shown above.
4. Save the changes.
5. In Cloudera Manager, select the Hive metastore service: Clusters HIVE_ON_TEZ-1 Configuration .
6. Search for compact.

HIVE_ON_TEZ-1 Actions

Status Instances **Configuration** Commands Charts Library Audits HiveServer2 Web UI Quick Links

Q compac Filters Role Groups

Filters

SCOPE

HIVE_ON_TEZ-1 (Service-Wide)	3
Gateway	0
HiveServer2	0

CATEGORY

Number of Threads Used by Compactor HIVE_ON_TEZ-1 (Service-Wide)

hive.compactor.worker.threads

Run compactor on Hive Metastore or HiveServer2. HIVE_ON_TEZ-1 (Service-Wide)

hive.metastore.runworker.in ☐ metastore ☒ hs2

7. Check that the Number of Threads Used by Compactor (hive.compactor.worker.threads), and Run compactor on Hive Metastore or HiveServer2 (hive.metastore.runworker.in) is set to hs2.
8. Save the changes and restart the Hive on Tez and Hive (HIVE-1) metastore services at an appropriate time.

Start 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

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

Start compaction using a query

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

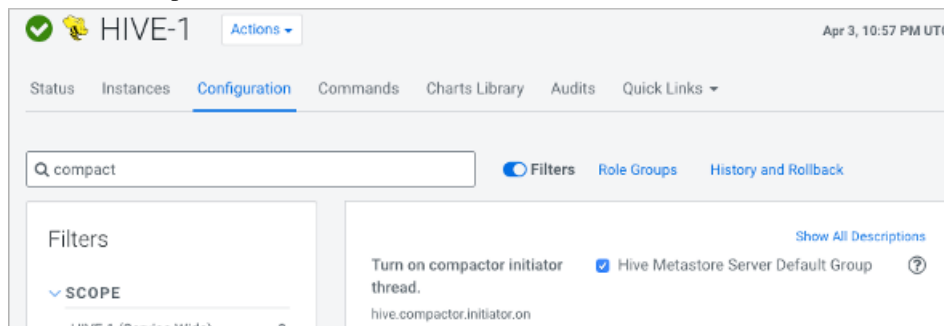
```
ALTER TABLE tablename [PARTITION (partition_key='partition_value' [...])]
COMPACT 'compaction_type'
```

Before you begin

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

Procedure

1. In Cloudera Manager, select the Hive metastore service: Clusters Hive-1 Configuration .
2. Search for compact.



3. Check that the Hive Metastore Server Default Group is selected (hive.compactor.initiator.on=true).
4. Execute a query to start a major compaction of a table.

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

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

View compaction progress

You view the progress of compactations by running a Hive query.

Procedure

Enter the query to view the progress of compactations.

SHOW COMPACTIONS;

- Unique internal ID
- Database name
- Table name
- Partition name
- Major or minor compaction

- Compaction state:
 - Initiated - waiting in queue
 - Working - currently compacting
 - Ready for cleaning - compaction completed and old files scheduled for removal
 - Failed - the job failed. Details are printed to the metastore log.
 - Succeeded
 - Attempted - initiator attempted to schedule a compaction but failed. Details are printed to the metastore log.
- Thread ID
- Start time of compaction
- Duration
- Job ID - ID of the submitted MapReduce job

Disable automatic compaction

You can disable automatic compaction of a particular Hive 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

Disabling automatic compaction does not prevent you from performing manual compaction.

Procedure

Start the Hive shell, and in the database of the target table, alter the TBLPROPERTIES.

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

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, valid values, and where to set these properties: Cloudera Manager, TBLPROPERTIES, hive-site.xml, or core-site.xml. When properties do not appear in Cloudera Manager search of configuration properties for a runtime service, you add the property to hive-site or core-site using the Cloudera Manager Safety Valve.

Basic compactor properties

hive.compactor.initiator.on

Default=false

Whether to run the initiator and cleaner threads on this metastore instance or not.

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 CDP 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

The number of aborted transactions involving a given table or partition that will trigger a major compaction.

Advanced compactor properties**hive.compactor.worker.timeout**

Default=86400s

Expects a time value 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 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 queue.

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 compaction history reaper runs.

Query vectorization

You can use vectorization to improve instruction pipelines and cache use. Vectorization enables certain data and queries to process batches of primitive types on entire column rather than one row at a time.

Default vectorized query execution

CDP enables vectorization by default and the value of `hive.vectorized.execution.enabled` is set to 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 contrary to the MapReduce execution process that stores temporary file.

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

Tracking Hive on Tez query execution

You need to know how to monitor Hive on Tez queries during execution. Several tools provide query details, such as execution time.

About this task

You can retrieve local fetch details about queries from HiveServer (HS2) logs, assuming you enable a fetch task. You configure the following properties:

hive.fetch.task.conversion

Value: minimal

Some select queries can be converted to a single FETCH task instead of a MapReduce task, minimizing latency. A value of none disables all conversion, minimal converts simple queries such as SELECT * and filter on partition columns, and more converts SELECT queries including FILTERS.

hive.fetch.task.conversion.threshold

Value: 1 GiB

Above this size, queries are converted to fetch tasks.

Increasing the static pool does not speed reads and there is not recommended.

Procedure

1. In Cloudera Manager, click Clusters Hive on Tez Configuration , and search for fetch.
2. Accept, or change, the default values of the fetch task properties.

The screenshot shows two configuration sections for 'Fetch Task Query Conversion' under the 'HiveServer2 Default Group'. The first section, labeled 'hive.fetch.task.conversion', has three radio button options: 'none', 'minimal', and 'more', with 'more' selected. The second section, labeled 'hive.fetch.task.conversion.threshold', has a text input field containing '1' and a unit dropdown menu set to 'GiB'. Both sections include an information icon (i) in the top right corner.

3. Navigate to the HiveServer log directory and look at the log files.

In Cloudera Manager, you can find the location of this directory as the value of HiveServer2 Log Directory.

The screenshot shows the 'HiveServer2 Log Directory' configuration under the 'HiveServer2 Default Group'. It features a text input field containing '/var/log/hive'. Below the field is a description: 'Directory where HiveServer2 will place its log files.' There is a close icon (X) in the top right corner.

4. In Cloudera Manager, click **Clusters** **Hive on Tez Configuration** , and click to the **HiveServer Web UI**.

HIVE_ON_TEZ-1 Actions ▾ Sep 13, 10:43 PM

[Status](#) [Instances](#) [Configuration](#) [Commands](#) [Charts Library](#) [Audits](#) [HiveServer2 Web UI](#) [Quick Links ▾](#)

HiveServer2

Active Sessions

User Name	IP Address	Operation Count	Active Time (s)	Idle Time (s)
admin	172.27.72.74	0	58523	58466

Total number of sessions: 1

Open Queries

User Name	Query	Execution Engine	State	Opened Timestamp	Opened (s)	Latency (s)	Drilldown Link
Total number of queries: 0							

Last Max 25 Closed Queries

User Name	Query
admin	INSERT INTO TABLE sample_07

5. Use Hue to track query progress.

Tracking an Apache Hive query in YARN

You need to know how to monitor Apache Hive queries in YARN. Using information from the output of query execution in Beeline, you can accomplish this task.

About this task

Procedure

1. Run a query in Beeline.

```
0: jdbc:hive2://ip-10-10-10-10.cloudera.site:> select * from depts;
INFO : Compiling command(queryId=hive_1599978188515_5723b397-c375-48c2-ab38-7be298603de9): select * from depts
...
```

2. Scroll down the output until you find the INFO containing the YARN App id.

```
...
INFO : Status: Running (Executing on YARN cluster with App id application_1599978188515_0010)
```

VERTICES			MODE	STATUS	TOTAL	COMPLETED	RUNNING	PE
ENDING	FAILED	KILLED						
...								

3. In Cloudera Manager, click Clusters Yarn Instances Web UI .

YARN-1

Actions

Status

Instances

Configuration

Commands

Applications

Charts Library

Audits

Web UI

Quick Links

ResourceManager Web UI (nightly7x-unsecure-3)

HistoryServer Web UI (nightly7x-unsecure-3)

4. Click Resource Web UI Applications

Application ID	Application Type	Application Tag	Application Name	User	State	Queue
application_1599978188515_0010	TEZ	N/A	HIVE-dcb03c1e-...	hive	Running	default
application_1599978188515_0009	SPARK	N/A	PySparkShell	hdfs	Finished	default

5. Find the match for the App id and gather information you want.

Application not running message

Understanding the Application not running message from TEZ that appears in the YARN application log prevents confusion when inspecting Hive queries.

During startup, HiveServer starts sessions that are available until tez.session.am.dag.submit.timeout.secs expires, and then the Application Masters are killed. HiveServer transparently restarts dead AMs on demand when you run another query. HiveServer tries to clean up sessions from the Tez pool during shutdown. "Application not running" message in a stack trace logging is not an issue. This message is just a trace logged when the session is closed or restarted and the AM has timed out.