Data Access 3

# Apache Hive Performance Tuning

**Date of Publish:** 2018-08-23


**http://docs.hortonworks.com**

# Contents

# Optimizing an Apache Hive data warehouse

You can tune your data warehouse infrastructure, components, and client connection parameters to improve the performance and relevance of business intelligence and other applications. Tuning Hive and background components that support Hive processing is particularly important as your workload and database volume increases.

Increasingly, enterprises want to run SQL workloads that return faster results than batch processing can provide. These enterprises often want data analytics applications to support interactive queries. Hive low-latency analytical processing (LLAP) can improve the performance of interactive queries. A Hive interactive query that runs on the Hortonworks Data Platform (HDP) meets low-latency, variably guaged benchmarks to which Hive LLAP responds in 15 seconds or fewer. LLAP enables application development and IT infrastructure to run queries that return real-time or near-real-time results.

You can further enhance LLAP performance with real-time data by integrating the enterprise data warehouse (EDW) with the Druid business intelligence engine.

When you query large-scale EDW data sets, you have to meet service-level agreement (SLA) benchmarks or other performance expectations. Because how you tune your query processing environment depends on factors such as system resources, depth of data analysis, and query latency requirements, you must become familiar with Hive warehouse processing, prepare for tuning, and configure LLAP using parameters that meet your performance needs.

# LLAP ports

You use port 10500 to make the JDBC connection through Beeline to query Hive through the HiveServer Interactive host. The LLAP daemon uses several other ports.

| |
|---|
| HiveServer Interactive (LLAP) port (10500) |
| hive.server2.thrift.http.port (10501) |
| hive.llap.daemon.rpc.port (0) |
| hive.llap.daemon.web.port (15002) |
| hive.llap.daemon.yarn.shuffle.port (15551) |
| hive.llap.management.rpc.port (15004) |

# Preparations for tuning performance

Before you tune Apache Hive, you should follow best practices. These guidelines include how you configure the cluster, store data, and write queries.

### Best practices

- Set up your cluster to use Apache Tez or the Hive on Tez execution engine.

  In HDP 3.x, the MapReduce execution engine is replaced by Tez.
- Disable user impersonation by setting Run as end user to false in Ambari, which is equivalent to setting hive.server2.enable.doAs in hive-site.xml.

  LLAP caches data for multiple queries and this capability does not support user impersonation.
- Add the Ranger security service to your cluster and dependent services.
- Set up LLAP to run interactive queries.

- Store data using the ORC File format.
- Ensure that queries are fully vectorized by examining explain plans.
- Use the SmartSense tool to detect common system misconfigurations.

# Setting up LLAP

Using Ambari, you can set up basic, low-latency analytical processing (LLAP) by accepting the default llap queue, changing some YARN queue properties, and adding a HiveServer property to hive-site.xml.

### Before you begin

- You have installed Ambari 2.7.x or later.
- You added and started YARN, Tez, and dependencies that Ambari prompts you to add.
- You added, started, and can run conventional queries from the Hive service.
- If enabled, you disabled maintenance mode for the Hive service and target host for HiveServer Interactive (HSI); otherwise, enabling LLAP fails to install HSI. Alternatively, you need to install HiveServer Interactive on the Ambari server as follows:

```
curl -u admin:<password> -H "X-Requested-By:ambari" -i -X POST
 http://host:8080/api/v1/clusters/<cluster_name>/hosts/<host_name>/
host_components/HIVE_SERVER_INTERACTIVE
```

### About this task

The Interactive Query control displays a range of values for default Maximum Total Concurrent Queries based on the number of nodes that you select for LLAP processing and the number of CPUs in the Hive LLAP cluster. The Ambari wizard typically calculates appropriate values for LLAP properties in Interactive Query, so you can usually accept the defaults.

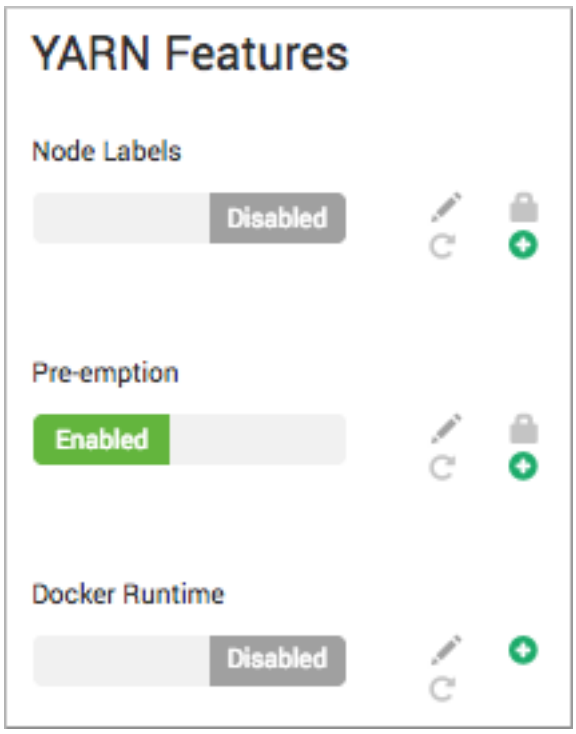
When you enable Interactive Query, the **Run as end user** and **Hive user** security settings have no effect. These controls affect batch-processing mode.

## Enable YARN preemption

You must enable YARN preemption before setting up Hive low-latency analytical processing (LLAP). YARN preemption directs the capacity scheduler to position an LLAP queue as the top-priority workload to run among cluster node resources. You do not need to label the nodes to ensure LLAP has top priority.

### Procedure

1. In Ambari, select **Services** > **YARN** > **Configs**.
2. In YARN Features, set Pre-emption to Enabled (the default).




3. Save the settings.

## Enable interactive query

You need to enable interactive query to take advantage of low-latency analytical processing (LLAP) of Hive queries.

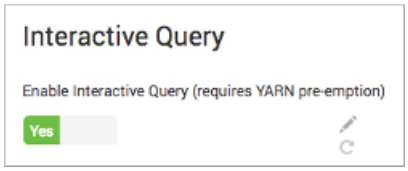
### About this task

The Interactive Query control displays a range of values for default Maximum Total Concurrent Queries based on the number of nodes that you select for LLAP processing and the number of CPUs in the Hive LLAP cluster. The Ambari wizard typically calculates appropriate values for LLAP properties in Interactive Query, so accept the defaults or change the values to suit your environment.

When you enable Interactive Query, the **Run as end user** and **Hive user** security settings have no effect. These controls affect batch-processing mode.

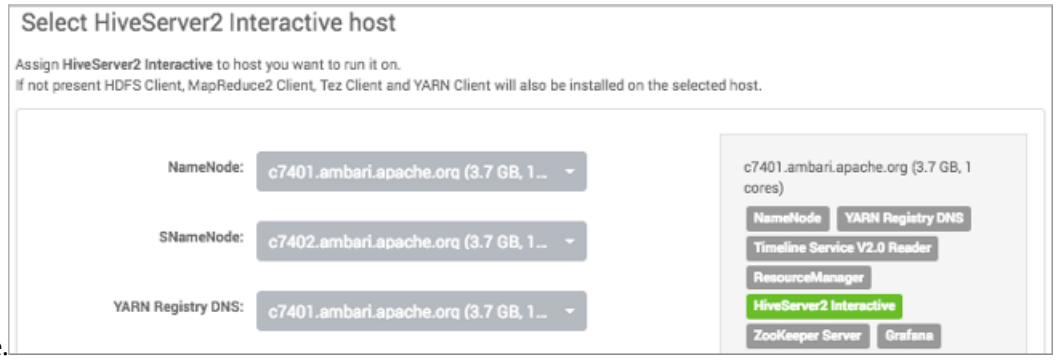
### Procedure

1. In Ambari, select **Services** > **Hive** > **Configs** > **Settings**.
2. In Interactive Query, set Enable Interactive Query to Yes:




3. In Select HiveServer Interactive Host, select the server to host HiveServer Interactive.

For example, accept the default highlighted server, which is typically a good choice.




## Configure an llap queue

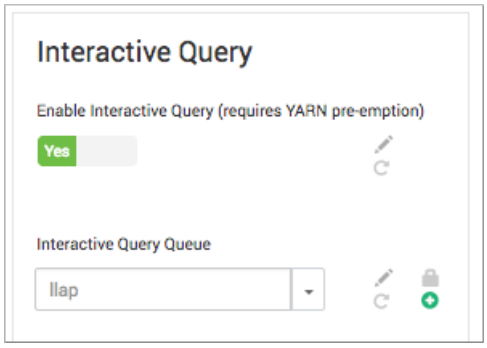
Ambari generally creates and configures an interactive query queue named llap and points the Hive service to a YARN queue. You check, and if necessary, change the llap queue using YARN Queue Manager.

### About this task

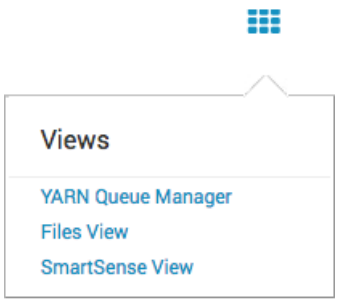
The llap queue capacity determines the YARN resources for the LLAP application. Reconfiguring the llap queue is sometimes necessary. For example, if you have a 3-node cluster, Ambari might configure zero percent capacity for the llap queue, and you must reconfigure settings. If you set the llap queue capacity or number of nodes too low, you won't have enough YARN resources or LLAP daemons to run the LLAP application. If you set the llap queue capacity too high, you waste space on the cluster.

### Procedure

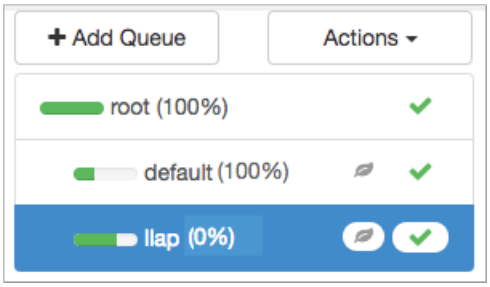
1. In Ambari, select **Hive** > **Configs**.
2. In Interactive Query Queue, choose the llap queue if it appears as a selection, and save the Hive configuration changes.:



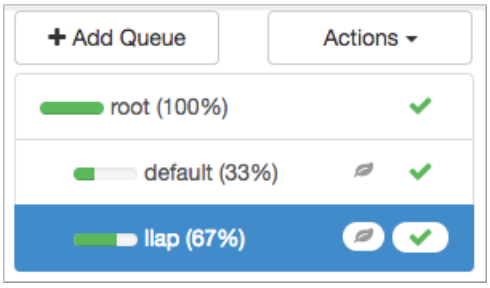

   Depending on your YARN Capacity Scheduler settings, a queue named llap might or might not appear. This setting dedicates all LLAP daemons and all YARN Application Masters (AMs) of the system to the single, specified queue.
3. In Ambari, select Services > YARN > Configs.
4. From the hamburger menu Views, select YARN Queue Manager.




5. If an llap queue does not exist, add a queue named llap. Otherwise, proceed to the next step.
6. If the llap queue is stopped, change the state to running.
7. Check the llap queue capacity, and accept or change the value as follows:
   - If the llap queue capacity is zero, you might have too few nodes for Ambari to configure llap queue capacity. Go to the next step to configure llap queue capacity and max capacity.

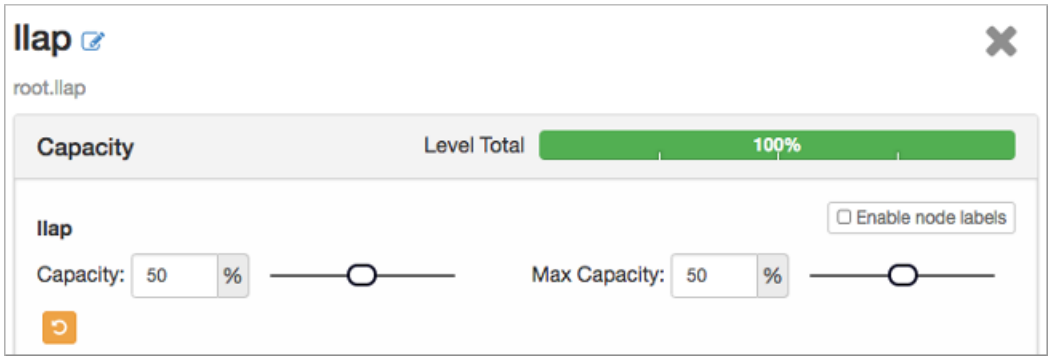



   - If Ambari set the llap capacity to greater than zero, no change is necessary. Skip the next step. For example, in a 7-node cluster, Ambari allocates llap queue capacity as follows:




8. If llap queue capacity is zero, increase the capacity allocated to your llap queue, and also change max capacity to the remainder of the allocated llap capacity minus 100 percent.

   For example, set max capacity to 100 percent minus 50 percent = 50 percent.




   Allocating 15-50 percent of cluster to the llap queue is common.
9. Select the llap queue under Add Queue, and in Resources that appears on the right, set User Limit Factor to 1, and set Priority to greater than 0 (1 for example).

10. Select **Actions** > **Save and Refresh Queues**.
11. In **Services** > **YARN** > **Summary** restart any YARN services as prompted.

## Add a Hive proxy

To prevent network connection or notification problems, you must add a hive user proxy for HiveServer Interactive service to access the Hive Metastore.

### Procedure

1. In Ambari, select **Services** > **HDFS** > **Configs** > **Advanced**.
2. In Custom core-site, add the FQDNs of the HiveServer Interactive host or hosts to the value of hadoop.proxyuser.hive.hosts.
3. Save the changes.

## Configure other LLAP properties

Configuring LLAP involves setting properties in YARN and Hive. After you configure the llap queue, you need to go back to the Hive configuration to continue setting up low-latency analytical processing (LLAP).

### About this task

In this task, you accept the Ambari configuration of a number of properties or reconfigure the properties. Ambari generally configures the number of nodes that run an LLAP daemon and total concurrent queries depending on the size of the llap queue. Ambari also attempts to correctly configure the following properties for your particular cluster:

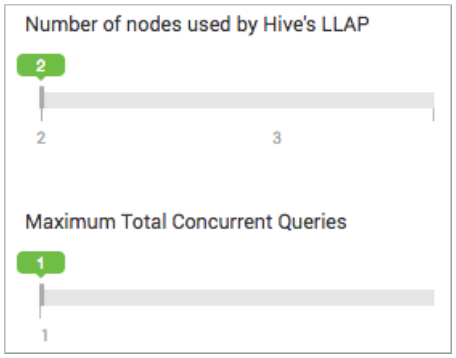
| | |
|---|---|
| **Memory per Daemon** | YARN container size for each daemon (MB) |
| **In-Memory Cache per Daemon** | Size of the cache in each container (MB) |
| **Number of executors per LLAP Daemon** | The number of fragments that can execute in parallel on a daemon |

Use the slider controls to change or restore settings:

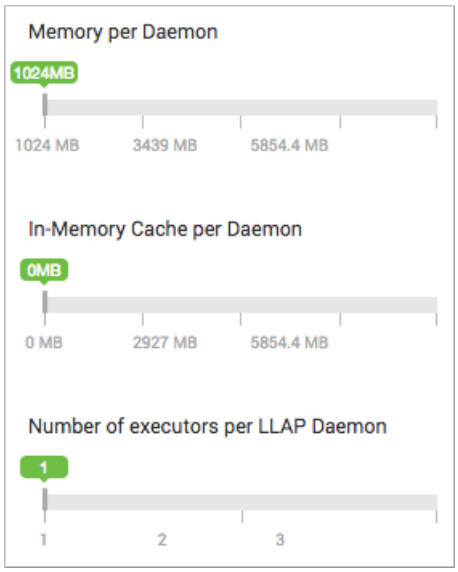

To set the value outside the slider range, you move your pointer over the field to enable the hover actions, and select Override.

### Procedure

1. Accept or change the Number of Nodes Used By Hive LLAP (num_llap_nodes property). For example, accept using 2 nodes for LLAP.

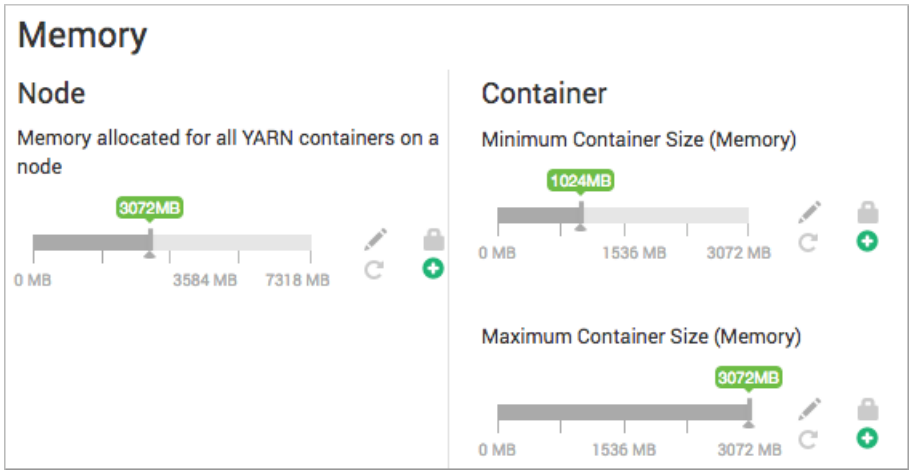



2. Accept the Maximum Total Concurrent Queries (hive.server2.tez.sessions.per.default.queue property), or make changes.
3. Check Memory per Daemon (hive.llap.daemon.yarn.container.mb property) and In-Memory Cache per Daemon (hive.llap.io.memory.size property).




   This memory (hive.llap.daemon.yarn.container.mb) plus the cache (hive.llap.io.memory.size) must fit within the container size specified for the YARN container. The YARN container configuration setting appears in **Services** > **YARN** > **Configs** > **Settings**.
4. Accept the Number of Executors per LLAP Daemon (hive.llap.daemon.num.executors), or change this setting if you know what you are doing, and check that the hive.llap.io.threadpool.size is the same value.
5. Save any Hive configuration changes, and in **Services** > **YARN** > **Settings** > **Memory - Node**, check that the Minimum Container Size (Memory) for YARN is low.

   The value should rarely exceed 1024 MB.
6. Set the Maximum Container Size (Memory) to the same value as the Memory Allocated for All YARN Containers on a Node.




7. In Ambari, select **Services** > **YARN** > **Configs** > **Advanced**.
8. In Custom yarn-site, add the following properties unless, upon attempting to add these properties, Ambari indicates the properties are already added:yarn.resourcemanager.monitor.capacity.preemption.natural_termination_factor (value = 1) and yarn.resourcemanager.monitor.capacity.preemption.total_preemption_per_round (as described below).

   Calculate the value of the total preemption per round by dividing 1 by the number of cluster nodes. Enter the value as a decimal.

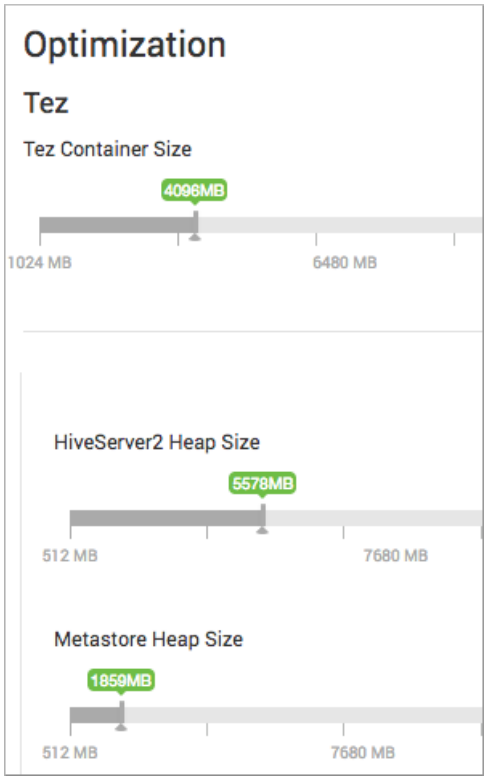
   For example, if your cluster has 20 nodes, then divide 1 by 20 and enter 0.05 as the value of this property setting.
9. Save YARN changes, and go back to the Hive configuration.

## Configure the HiveServer heap size

Generally, configuring HiveServer heap memory according to the recommendations in this topic ensures proper LLAP functioning. If HiveServer heap memory is set too low, HiveServer Interactive keeps going down.

### Procedure

1. In Ambari, go to **Services** > **Hive** > **Config**.
2. In Optimization, adjust the slider to set the heap size.



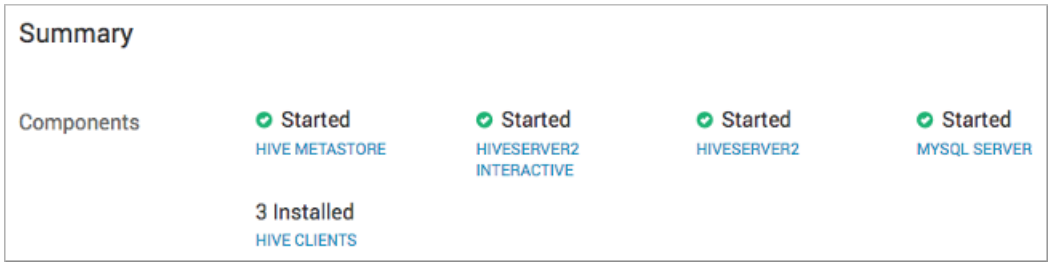

For 1 to 20 concurrent executing queries, set to 6 GB heap size; 21 to 40 concurrent executing queries: Set to 12 GB heap size.

## Save LLAP settings and restart services

You need to save LLAP settings at the bottom of the Ambari wizard and restart services in the proper order to activate low-latency analytical processing (LLAP).

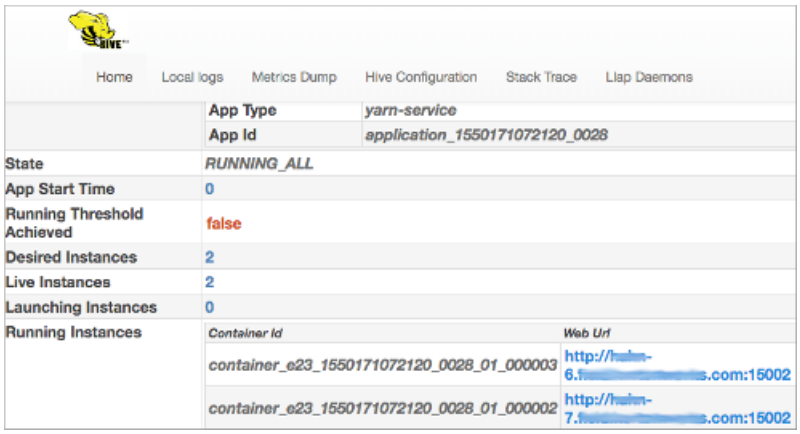
### Procedure

1. Click Save at the bottom of the wizard.
2. If the Dependent Configurations window appears, review recommendations and accept or reject the recommendations.
3. Navigate to the each service, starting with the first one listed under Ambari Services, and restart any services as required.
4. Select **Services** > **Hive** > **Summary** and verify that HiveServer Interactive is set up and started.

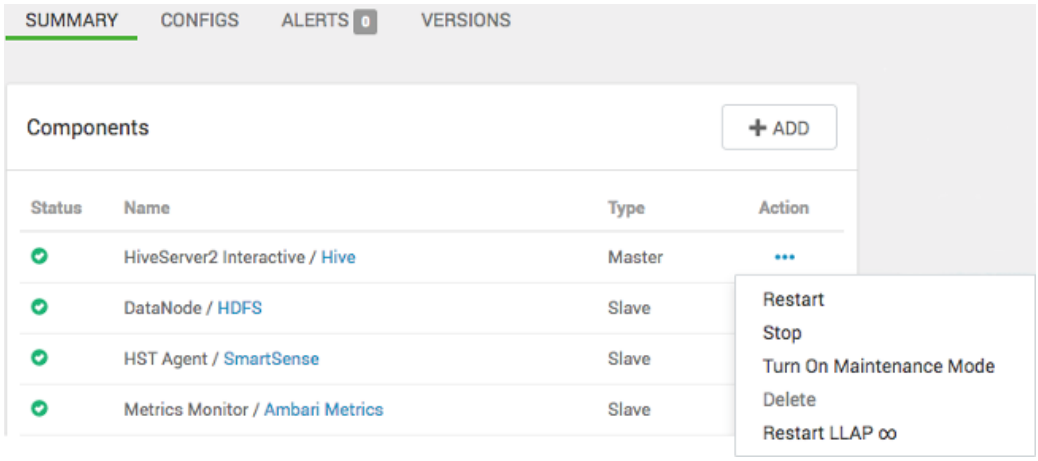



5. In Ambari, select **Services** > **Hive** > **Summary** and in Quick Links, click HiveServer Interactive UI to check that the LLAP application is running.


The HiveServer Interactive UI shows LLAP running on two daemons (instances).




6. If LLAP is not running, click HiveServer Interactive in Summary, and then, choose Restart LLAP from the Action menu.

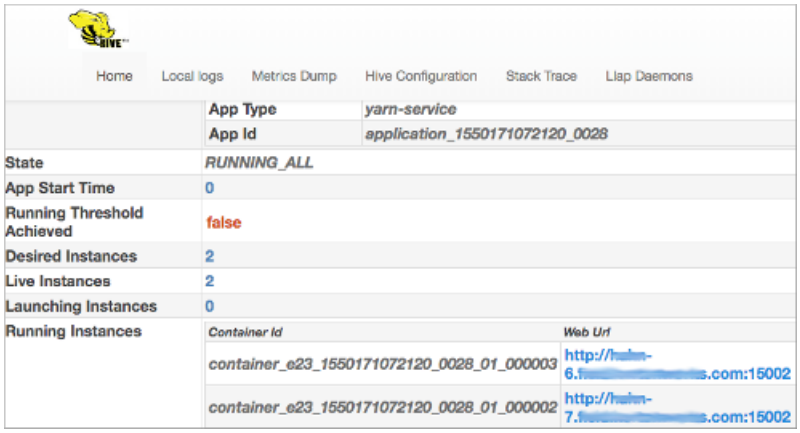



## Run an interactive query

You connect to HiveServer through Beeline to run interactive queries, which are queries that take advantage of low-latency analytical processing (LLAP). In the connection string, you specify the FQDN of the node that runs HiveServer Interactive.

### Before you begin

- You set up LLAP and restarted services.
- You checked the HiveServer Interactive UI, which you access from **Summary** > **Quick Links** > **HiveServer Interactive UI**, and you see that LLAP is running.




### Procedure

1. On the command-line of a node in the cluster, connect to HiveServer Interactive on port 10500 through Beeline.

   For example, enter the following beeline command, but replace my_hiveserver_interactive.com with the FQDN of your HiveServer Interactive node:

   ```
   $ beeline -n hive -u jdbc:hive2://
   my_hiveserver_interactive.com:10500/;transportMode=binary
   ```

2. At the Hive prompt, create a table and insert data.

   ```
   CREATE TABLE students (name VARCHAR(64), age INT, gpa DECIMAL(3,2));
   INSERT INTO TABLE students VALUES ('fred flintstone', 35, 1.28), ('barney
    rubble', 32, 2.32);
   ```

   You probably notice that Hive inserted the data much faster using the LLAP interactive query than using a conventional Hive query.

# LLAP and HiveServer Interactive properties

Using Ambari **Services** > **Hive** > **Settings**, you can check and configure key properties in Interactive Query settings. You can check, configure, or add some LLAP properties in **Hive** > **Services** > **Configs** > **Advanced**. Knowing the internal names and definitions of the properties helps simplify the LLAP configuration task.

| Ambari UI Component | Property | Access From | Description |
|---|---|---|---|
| Custom hiveserver2-interactive-site | hive.server2.tez.interactive.queue | Services > Hive > Configs | The name of a YARN queue for workload management, required for managing workloads |
| In-Memory Cache per Daemon | hive.llap.io.memory.size | Interactive Query | Memory reserved for in-memory cache (memory per daemon - heap – headroom). |
| Memory per Daemon | hive.llap.daemon.yarn.container.mb | Interactive Query | Total memory used by LLAP daemons (YARN Container size), which includes memory for daemon cache and query execution (cache + heap + headroom = memory per daemon.) Recommendation: Set to yarn.scheduler.maximum-allocation-mb |

| Ambari UI Component | Property | Access From | Description |
|---|---|---|---|
| Tez Container Size | hive.tez.container.size | Optimization | Change to override java options from map tasks that Tez uses by default. Recommended: 4 – 6 GB recommended, for each executor allocate one VCPU |
| LLAP Daemon Container Max Headroom | llap_headroom_space | Advanced hive-interactive-env | Maximum headroom reserved from YARN container running LLAP. The upper limit used for automatic size calculations. Actual value might be lower. Minimum: 5 percent of LLAP Daemon Size or 6 GB. |
| LLAP Daemon Heap Size (MB) | llap_heap_size | Advanced hive-interactive-env | hive.llap.daemon.num.executors * hive.tez.container.size |
| LLAP Queue Size | yarn.scheduler.capacity.root.llap.capacity | Capacity Scheduler | The capacity as a percentage of total allocated to the queue. |
| Maximum Total Concurrent Queries | hive.server2.tez.sessions.per.default.queue | Services > Hive > Configs > Advanced | A positive integer that determines the number of Tez sessions (parallelism) launched on each hive.server2.tez.default.queue. Spawns an equal number of TEZ AMs. |
| Maximum Container Size (Memory) | yarn.scheduler.maximum-allocation-mb | Services > YARN > Configs > Memory | The maximum allocation for every container request at the Resource Manager in MBs. Higher memory capped to this value. |
| Number of Nodes for Running Hive LLAP Daemon | num_llap_nodes_for_llap_daemons | Advanced hive-interactive-env | Self-explanatory |
| Number of Executors per LLAP Daemon | hive.llap.daemon.num.executors | Interactive Query | The number of fragments a single LLAP daemon runs concurrently, generally equals yarn.scheduler.maximum-allocation-vcores. |
| Tez AM coordinator Size | tez.am.resource.memory.mb | Advanced tez-interactive-site | The memory used by the AppMaster only if the value is not explicit in the DAG definition. Same as yarn.scheduler.minimum-allocation-mb. |

### Related Information

Configure a YARN queue for workload management

# Use HiveServer Interactive UI

You can access the HiveServer Interactive UI in Ambari to monitor and display heap, system, and cache metrics of each Hive LLAP node.

### Before you begin

- You installed Hive using Ambari and Hive is running.
- You enabled LLAP and started HiveServer Interactive.

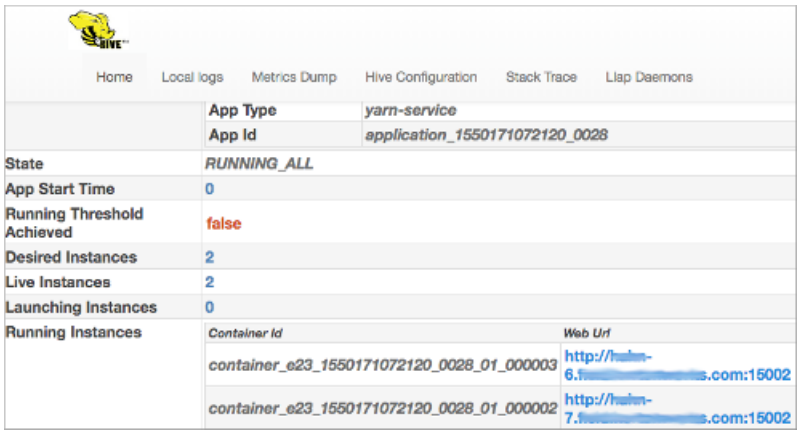
### About this task

In this task you open the HiveServer Interactive UI and view executing queries in real time or a recent history of queries. From the UI, you can also access running LLAP daemons.

### Procedure

1. In Ambari, select **Services** > **Hive** > **Summary**and in Quick Links, click HiveServer Interactive UI.


   The HiveServer Interactive UI shows LLAP running on two daemons (instances).




2. Click Local Logs.
   A list of HiveServer Interactive logs appears.
3. Click hiveserver2Interactive.log.
   The logged data from /var/log/hive/hiveserver2Interactive, which you can use for troubleshooting HiveServer Interactive problems, appears.
4. Click hive-server2-interactive.err.
   The logged data about queries, which includes query and task execution summaries, appears.

# Connect a JDBC client to LLAP

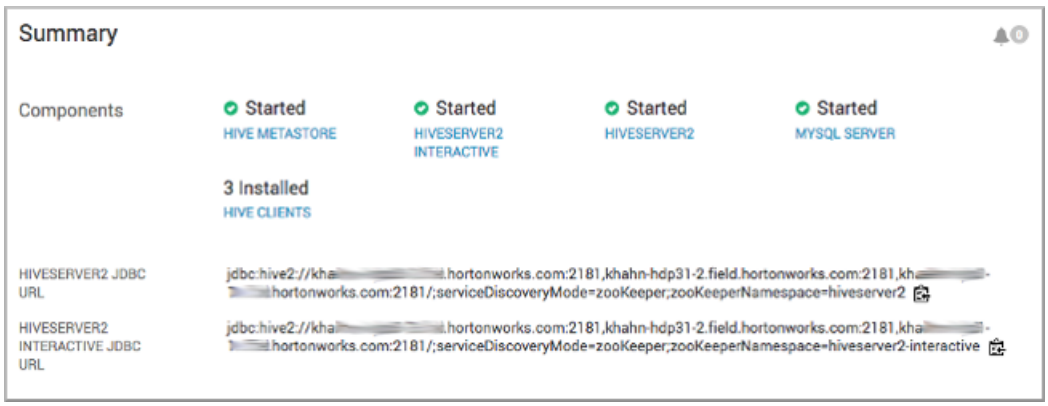
As administrator, you can provide users with the JDBC URL generated by HiveServer, so they can issue queries to the Hive LLAP instance.

### Before you begin

- You completed the setup of Hive LLAP in Ambari and restarted the Hive service.

### Procedure

1. In Ambari, select Services > Hive .
2. In Summary, copy the JDBC URL for HiveServer Interactive: Click the clipboard icon.




3. In your JDBC client, paste the JDBC URL.
   Now, you can connect a JDBC client to Hive LLAP and run queries.
4. Provide the JDBC standalone JAR by downloading it from **Services** > **Hive** > **Summary** > **Quick Links**.


# Configuring YARN queues for Hive

To perform Hive batch processing, you need to set up a YARN queue. As an advanced user, you might want to set up a custom LLAP queue instead of using the default llap queue that Ambari creates.

## Configure a queue for batch processing

You can configure the capacity scheduler queues to scale a Hive batch job for your environment. YARN uses the queues to allocate Hadoop cluster resources among users and groups.

### About this task

In this task, you create queues and set up a capacity scheduler to separate short- and long-running queries into the queues:

| | |
|---|---|
| **hive1** | This queue is used for short-duration queries and is assigned 50 percent of cluster resources. |
| **hive2** | This queue is used for longer-duration queries and is assigned 50 percent of cluster resources. |

### Procedure

1. In Ambari, access the capacity scheduler:
   - Select **YARN** > **Configs** > **Advanced**, and in Filter enter yarn.scheduler.capacity.root.
   - On the command line of the node where YARN is installed, go to the YARN /conf file, and open the capacity-scheduler.xml file.
2. Define the hive1 and hive2 queues, and set the maximum capacity to 50 percent of the queue users with a hard limit.

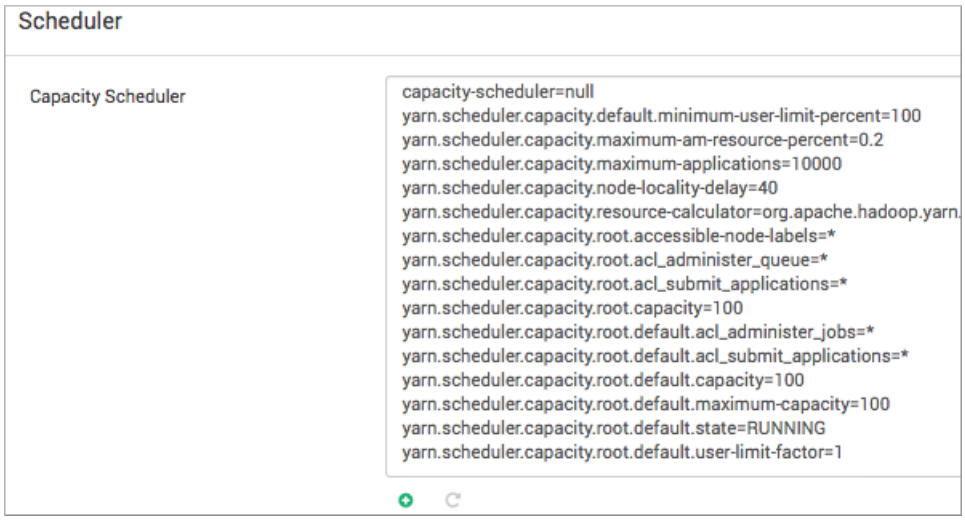
For example:

```
yarn.scheduler.capacity.root.queues=hive1,hive2
yarn.scheduler.capacity.root.hive1.capacity=50
yarn.scheduler.capacity.root.hive2.capacity=50
```

If the maximum-capacity is set to more than 50 percent, the queue can use more than its capacity when there are other idle resources in the cluster.

3. Configure usage limits for these queues and their users.
For example:

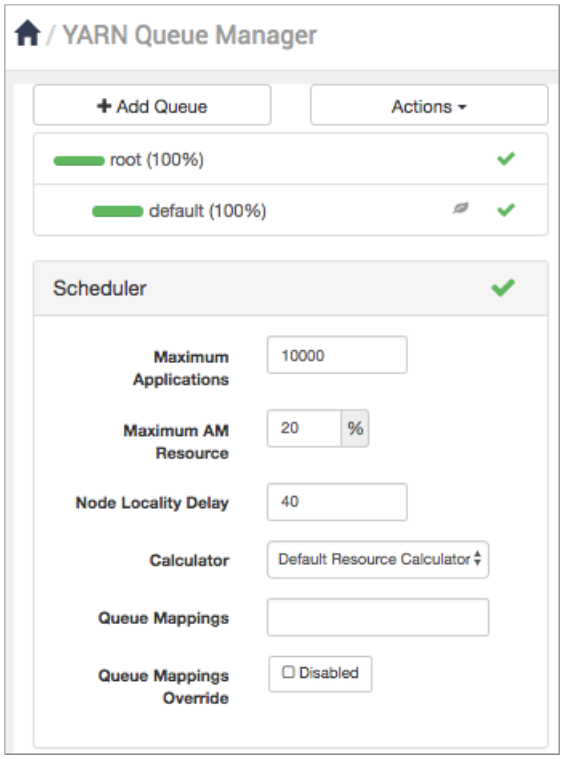
```
yarn.scheduler.capacity.root.hive1.maximum-capacity=50
yarn.scheduler.capacity.root.hive2.maximum-capacity=50
yarn.scheduler.capacity.root.hive1.user-limit=1
yarn.scheduler.capacity.root.hive2.user-limit=1
```



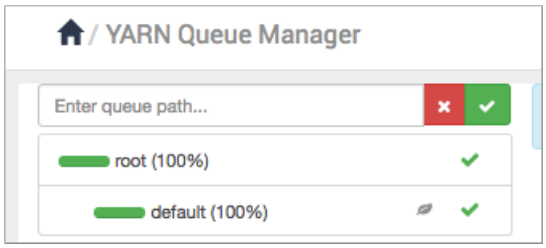

The default value of 1 for user-limit means that any single user in the queue can at a maximum occupy 1X the queue's configured capacity. These settings prevent users in one queue from monopolizing resources across all queues in a cluster.

4. From the Ambari dashboard, select **ADMIN** > **Manage Ambari** > **Views**.

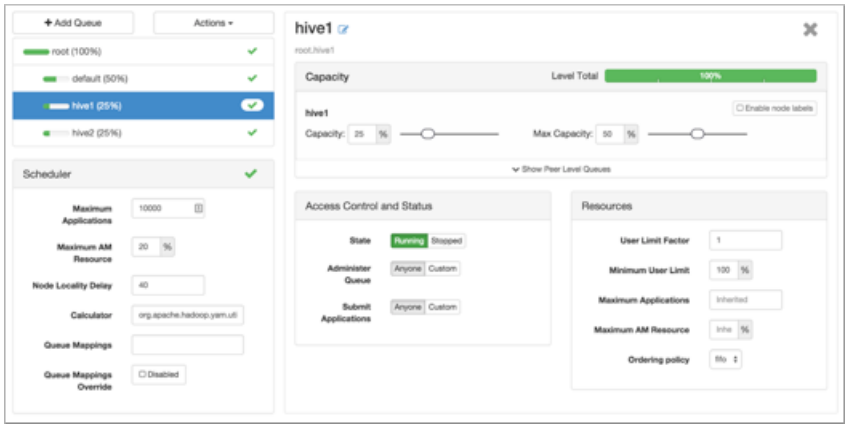

5. Click the URL for the view named AUTO_CS_INSTANCE, which is the capacity scheduler view.




6. In the YARN Queue Manager, click Add Queue.




7. Enter the queue path, which is the name of the first queue hive1, and then add the hive2 queue.
8. To create the following schedule, select the root queue and add hive1 and hive2 at that level:




## Configure a custom LLAP queue

As an advanced user, you can create and customize a queue for interactive querying instead of accepting the default llap queue during LLAP setup in Ambari. Using custom queues gives you the flexibility to assign different priorities, or other configurations, to queries run on certain queues, for example.

### Before you begin

You have not set up the default llap queue in Ambari.

### About this task

This task is optional. Do not set up a custom LLAP queue unless you are an experienced user.

### Procedure

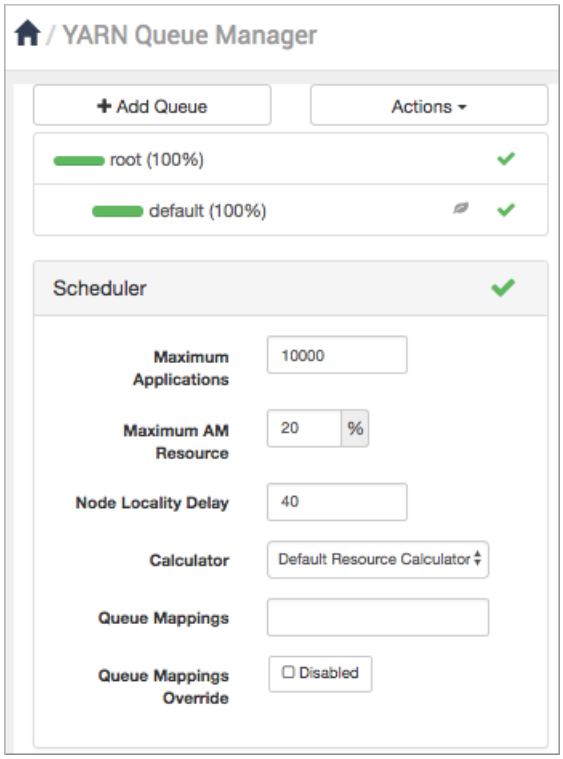
1. In Ambari, access the capacity scheduler:
   - Select **YARN** > **Configs** > **Advanced**, and in Filter enter yarn.scheduler.capacity.root.
   - On the command line of the node where YARN is installed, go to the YARN /conf file, and open the capacity-scheduler.xml file.
2. Define the custom LLAP queue, specifying a name and other properties, such as capacity and usage limits. For example:

```
yarn.scheduler.capacity.root.queues=myllap
yarn.scheduler.capacity.root.myllap=50
yarn.scheduler.capacity.root.myllap.maximum-capacity=50
yarn.scheduler.capacity.root.myllap.user-limit=1
```

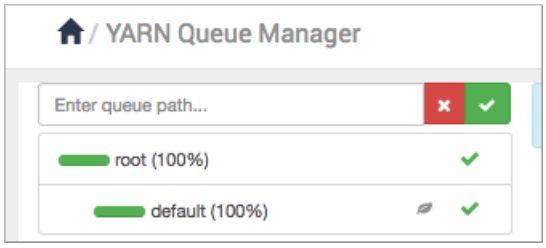
3. From the Ambari dashboard, select **ADMIN** > **Manage Ambari** > **Views**.

| Name | URL | View Type |
|---|---|---|
| AUTO_CS_INSTANCE | /main/view/CAPACITY-SCHEDULER/auto_cs_instance | CAPACITY-SCHEDULER {1.0.0} |
| AUTO_FILES_INSTANCE | /main/view/FILES/auto_files_instance | FILES {1.0.0} |
| SMARTSENSE_AUTO_INSTANCE | /main/view/SMARTSENSE/smartsense_auto_instance | SMARTSENSE {1.5.0.2.7.0.0-703} |

4. Click the URL for the view named AUTO_CS_INSTANCE, which is the capacity scheduler view.

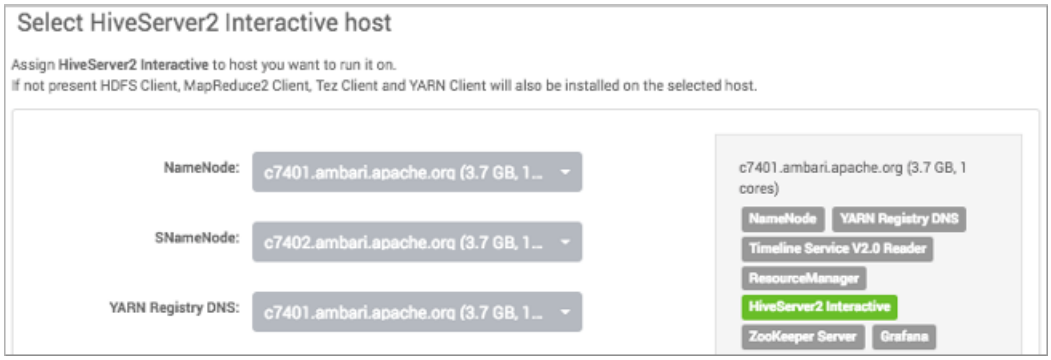



5. In the YARN Queue Manager, click Add Queue.

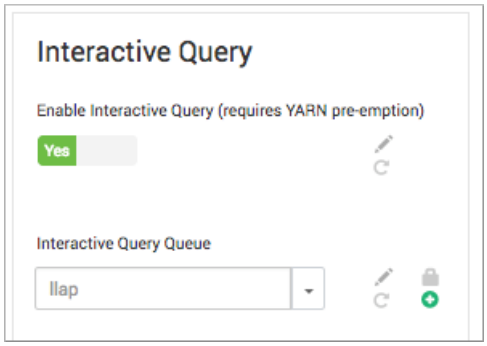



6. Enter the queue path, which is the name of the custom queue myllap hive1.
7. In Ambari, select **Services** > **Hive** > **Configs** > **Settings**.
8. In Interactive Query, set Enable Interactive Query to Yes:


9. In Select HiveServer Interactive Host, select the server to host HiveServer Interactive.
For example, select the default highlighted server, which is typically a good choice.




10. In Interactive Query Queue, select the custom YARN queue to replace the default llap:




This action dedicates all the LLAP daemons and YARN ApplicationMasters of the system to the single, specified queue.

# Set up multiple HiveServer instances

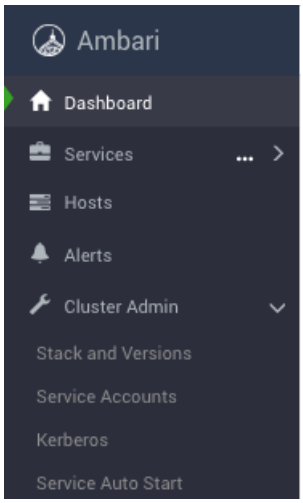
You can set up HiveServer instances, which can share the same metastore database for different purposes, such as ETL, running multiple applications using different settings, or load-balancing using ZooKeeper.

### Before you begin

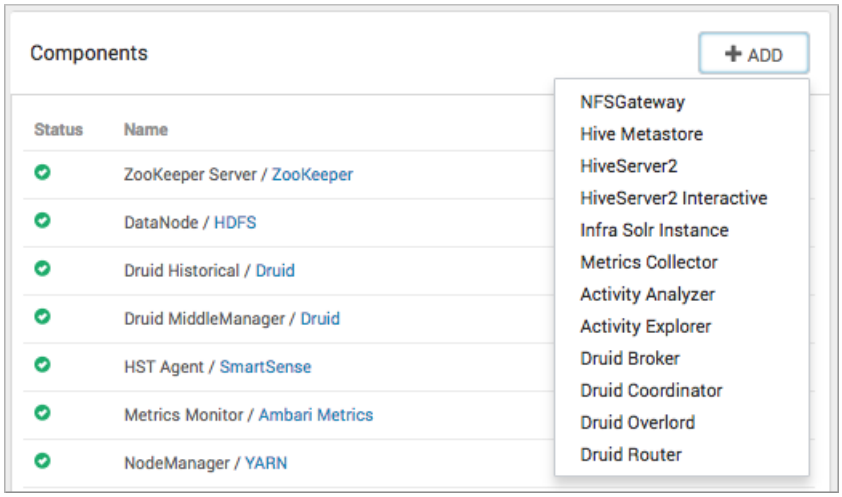
- You installed Hive using Ambari and Hive is running.

### Procedure

1. In Ambari, select Hosts.




2. Click the name of the host node where you want to create the HiveServer instance.
3. In Summary, in Components, click Add, and select HiveServer.




Do not add HiveServer Interactives for high availability using this procedure.

# Key components of Hive warehouse processing

A brief introduction to these software and systems, which affect performance, helps you understand the scope of the tuning task: HiveServer, admission control, batch processing using Apache Tez, interactive processing using Apache Tez and LLAP daemons, and YARN.

### HiveServer

HiveServer provides Hive service to multiple clients that simultaneously execute queries against Hive using an open Apache Hive API driver, such as JDBC. For optimal performance, you should use HiveServer to connect your client application and the Hive enterprise data warehouse (EDW). Using Ambari, you can install, configure, and monitor the Hive service and HiveServer.

An embedded metastore, which is different from MetastoreDB, runs in HiveServer and performs the following tasks:

- Gets statistics and schema from MetastoreDB
- Compiles queries

- Generates query execution plans
- Submits query execution plans
- Returns query results to the client

### Admission control

HiveServer coordinates admission control. Admission control is critical to Hive performance tuning. Admission control manages queries in a manner similar to how connection pooling manages network connections in RDBMS databases. When using the Hive LLAP on the Tez engine, you can configure admission control.

Admission control performs the following functions:

- Enables optimal Hive performance when multiple user sessions generate asynchronous threads simultaneously
- Scales concurrent queries to suit system resources and demand
- Postpones processing or cancels other queries if necessary

### Batch processing using Apache Tez

Each queue must have the capacity to support one complete Tez application, as defined by its Application Master (AM) (tez.am.resource.memory.mb property). Consequently, the number of Apache Tez AMs limits the maximum number of queries a cluster can run concurrently. A Hive-based analytic application relies on YARN containers as execution resources. The Hive configuration defines containers. The number and longevity of containers that reside in your environment depend on whether you want to run with batch workloads or enable Hive LLAP in HDP.

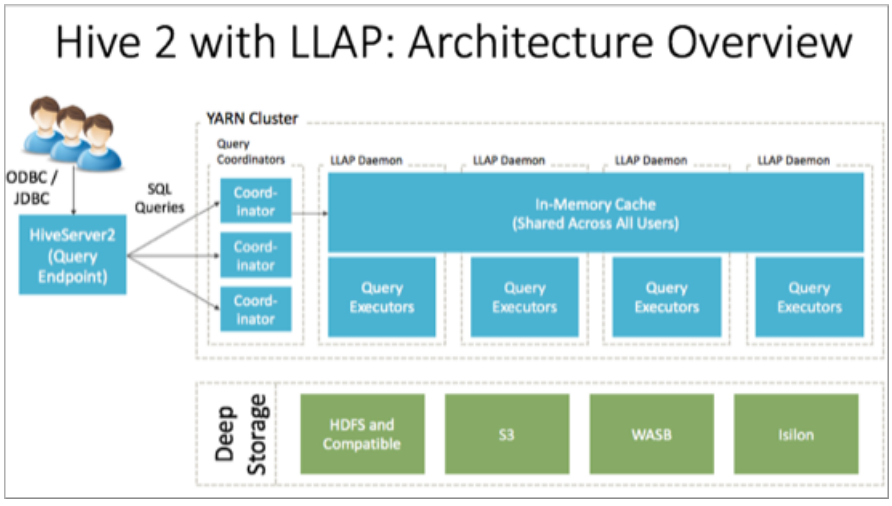
Hive on Tez is an advancement over earlier application frameworks for Hadoop data processing, such as using Hive on MapReduce, which is not supported in HDP 3.0 and later. The Tez framework is suitable for high-performance batch workloads.

After query compilation, HiveServer generates a Tez graph that is submitted to YARN. A Tez AM monitors the query as it runs.

### Interactive processing using Apache Tez and LLAP daemons

The way interactive and batch workloads operate with YARN and queues differs. Hive LLAP manages resources globally, rather than managing resources of each Tez session independently. LLAP does not rely on YARN: LLAP has its own resource scheduling and pre-emption built in. Consequently, you need only a single queue to manage all LLAP resources. Each LLAP daemon runs as a single YARN container.

The following diagram shows the architecture of Hive LLAP:




**Query coordinators**

Components that coordinate the execution of a single, persistent LLAP daemon, typically one per node. This is the main differentiating component of the

| | |
|---|---|
| | architecture, which enables faster query runtimes than earlier execution engines. |
| **Query executors** | Threads running inside the LLAP daemon. |
| **In-memory cache** | A cache inside the LLAP daemon that is shared across all users. |

### YARN

LLAP depends on YARN queues. HiveServer and YARN work together to intelligently queue incoming queries of a Hive data set. The queuing process tends to minimize the latency of returned results. The LLAP daemons manage resources across YARN nodes. When you set up LLAP, you enable interactive queries, enable YARN pre-emption, and choose a YARN queue for LLAP processing.

# Query result cache and metastore cache

You need to understand the query result cache to enable or disable it for debugging and to configure the memory allocated for the results. You also need to understand how the metadata cache affects data consistency.

### Using the query result cache

Some operations support hundreds of thousands of users who connect to Hive using BI systems such as Tableau. In these situations, repetitious queries are inevitable. The query result cache, which is on by default, filters and stores common queries in a cache. When you issue the query again, Hive retrieves the query result from a cache instead of recomputing the result, which takes a load off the backend system.

Every query that runs in Hive 3 stores its result in a cache. Hive evicts invalid data from the cache if the input table changes. For example, if you preform aggregation and the base table changes, queries you run most frequently stay in cache, but stale queries are evicted. The query result cache works with managed tables only because Hive cannot track changes to an external table. If you join external and managed tables, Hive falls back to executing the full query. The query result cache works with ACID tables. If you update an ACID table, Hive reruns the query automatically.

You can enable and disable the query result cache from command line. You might want to do so to debug a query. You disable the query result cache by setting the following parameter to false: hive.query.results.cache.enabled=false

Hive stores the query result cache in /tmp/hive/__resultcache__/. By default, Hive allocates 2GB for the query result cache. You can change this setting by configuring the following parameter in bytes: hive.query.results.cache.max.size

### Using the metadata cache

When query execution time is less than 1 second, compilation time dominates. Metadata retrieval is often a significant part of compilation time, which is devoted to RDBMS queries. In this situation, Cloud RDBMS As a Service is often slower, and frequent queries leads to throttling. Metadata caching speeds compilation time by approximately 50 percent in an on-premises MySQL installation. Significantly more improvements occur with cloud RDBMS. Caching achieves consistency in a single metastore setup and eventual consistent in an HA setup.

# Tez execution engine properties

If the performance of low-latency analytical processing (LLAP) degrades, you can adjust Tez properties to tune Hive LLAP performance, but generally changing Tez properties is not necessary.

### Tuning Tez

In Ambari, select **Services** > **Tez** > **Configs**to change general or advanced properties using guidelines in the following table. You can search for a property by entering it in the Filter field.

| Property | Guideline | Default |
|---|---|---|
| tez.am.resource.memory.mb | 4 GB maximum for most sites. Generally, do not change the value of this property unless you change resources on the cluster. | Depends on your environment |
| tez.session.am.dag.submit.timeout.secs | 300 minimum | 300 |
| tez.am.container.idle.release-timeout-min.millis | 20000 minimum | 10000 |
| tez.am.container.idle.release-timeout-max.millis | 40000 minimum | 20000 |
| tez.shuffle-vertex-manager.desired-task-input-size | Increase for large ETL jobs that run too long. | No default value set |
| tez.min.partition.factor | Increase for more reducers. Decrease for fewer reducers. | 0.25 |
| tez.max.partition.factor | Increase for more reducers. Decrease for fewer reducers. | 2.0 |
| tez.shuffle-vertex-manager.min-task-parallelism | Set a value if reducer counts are too low, even if the tez.shuffle-vertex-manager.min-src-fraction property is already adjusted. | No default value set |
| tez.shuffle-vertex-manager.min-src-fraction | Increase to start reducers later. Decrease to start reducers sooner. | 0.2 |
| tez.shuffle-vertex-manager.max-src-fraction | Increase to start reducers later. Decrease to start reducers sooner. | 0.4 |
| hive.vectorized.execution.enabled | true | 0.4 |
| hive.mapjoin.hybridgrace.hashtable | true for slower but safer processing. false for faster processing. | false |

# Monitoring Apache Hive performance

You can use Grafana, part of Ambari Metrics, to monitor Hive performance. Grafana includes prebuilt dashboards for advanced visualization of cluster metrics. From these dashboards, you can assess the performance of the system.

Grafana includes the following Hive LLAP dashboards:

- Hive LLAP Heatmap

  Shows all the nodes that are running LLAP daemons with percentage summaries for available executors and cache. This dashboard enables you to identify the hotspots in the cluster in terms of executors and cache.
- Hive LLAP Overview

  Shows the aggregated information across all of the clusters: for example, the total cache memory from all the nodes. This dashboard enables you to see that your cluster is configured and running correctly. For example, you might have configured 10 nodes, but see executors and cache indicators for only 8 nodes.
- Hive LLAP Daemon

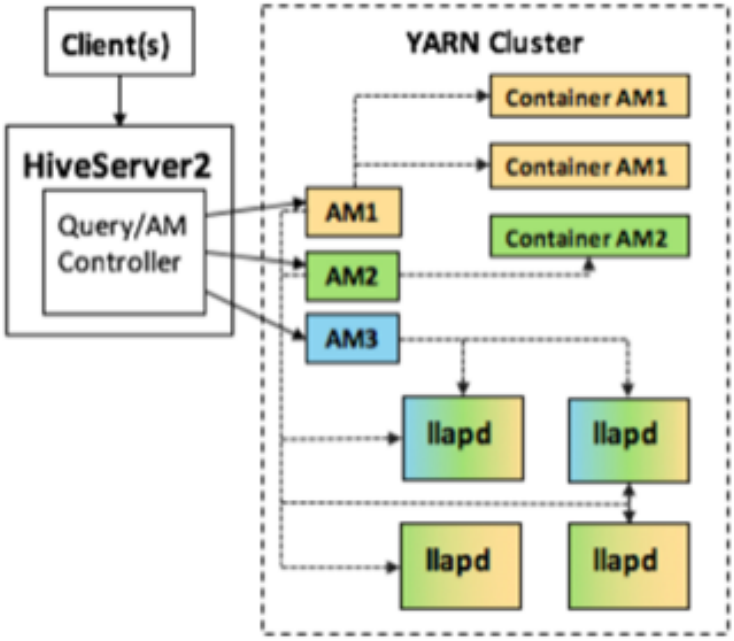
  Metrics that show the operating status for a specific Hive LLAP Daemon.

From an issue in the Hive LLAP Overview dashboard, you can go to the LLAP Daemon dashboard and determine which node is having the problem.

## Monitoring LLAP resources

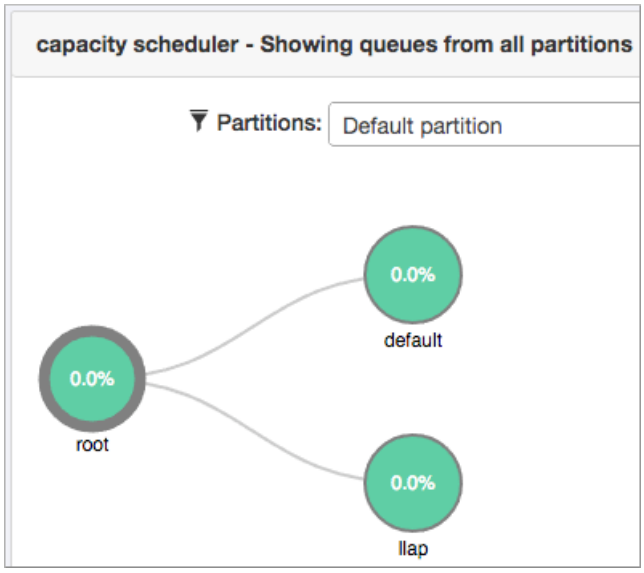
As administrator, you can monitor a cluster to conserve resources when running low-latency analytical processing (LLAP). You can monitor the queries in real-time using the YARN ResourceManager Web UI or YARN command-line tools. You can view LLAP app information and HiveServer Interactive logs using the HiveServer2 Interactive UI.

After setup, Hive LLAP is transparent to Apache Hive users and business intelligence tools. Interactive queries run on Apache Hadoop YARN. For example, you can run a large Hive LLAP cluster during the day for BI tools, and then reduce usage during nonbusiness hours to use the cluster resources for ETL processing. The following diagram shows how LLAP fits into your cluster:

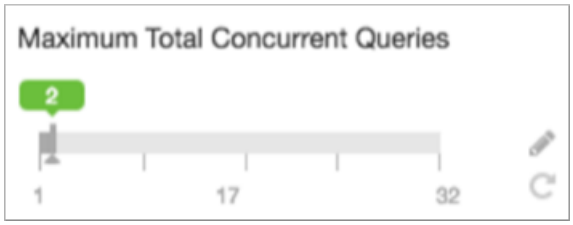



On your cluster, an extra HiveServer instance is installed that is dedicated to interactive queries. You can see this HiveServer instance listed in the Hive Summary page of Ambari.

In the YARN ResourceManager UI, you can see the queue of Hive LLAP daemons or running queries:



The Apache Tez ApplicationMasters are the same as the selected concurrency. If you selected a total concurrency of 5, you see 5 Tez ApplicationMasters. The following example shows selecting a concurrency of 2:

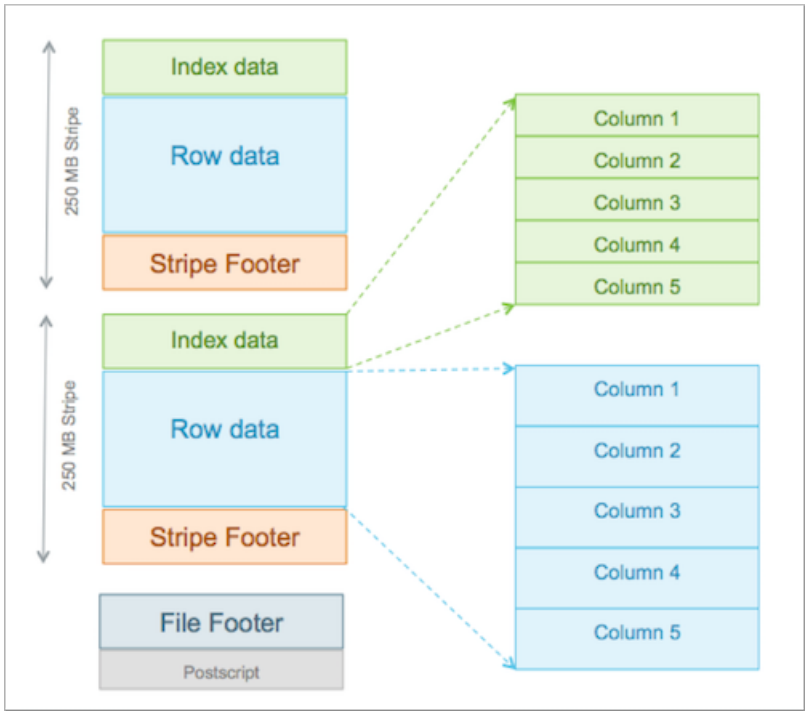



# Maximizing storage resources using ORC

You can conserve storage in a number of ways, but using the Optimized Row Columnar (ORC) file format is most effective.

The ORC file format for data storage is recommended for the following reasons:

- Efficient compression: Stored as columns and compressed, which leads to smaller disk reads. The columnar format is also ideal for vectorization optimizations in Tez.
- Fast reads: ORC has a built-in index, min/max values, and other aggregates that cause entire stripes to be skipped during reads. In addition, predicate pushdown pushes filters into reads so that minimal rows are read. And Bloom filters further reduce the number of rows that are returned.
- Proven in large-scale deployments: Facebook uses the ORC file format for a 300+ PB deployment.




ORC provides the best Hive performance overall. In addition, to specifying the storage format, you can also specify a compression algorithm for the table, as shown in the following example:

```
CREATE TABLE addresses (
 name string,
 street string,
 city string,
 state string,
 zip int
```

```
 ) STORED AS orc TBLPROPERTIES ("orc.compress"="Zlib");
```

Setting the compression algorithm is usually not required because your Hive settings include a default algorithm. Using ORC advanced properties, you can create bloom filters for columns frequently used in point lookups.

You can read a table and create a copy in ORC using the following command:

```
CREATE TABLE a_orc STORED AS ORC AS SELECT * FROM A;
```

A common practice is to store data in HDFS as text, create a Hive external table over it, and then store the data as ORC inside Hive where it becomes a Hive-managed table.

Hive supports Parquet and other formats. You can also write your own SerDes (Serializers, Deserializers) interface to support custom file formats.

### Storage layer

While a Hive enterprise data warehouse (EDW) can run on one of a variety of storage layers, HDFS and Amazon S3 are among the most frequently used for data analytics in the Hadoop stack. Amazon S3 is a commonly used for a public cloud infrastructure.

A Hive EDW can store data on other file systems, including WASB and ADLS.

Depending on your environment, you can tune the file system to optimize Hive performance by configuring compression format, stripe size, partitions, and buckets.

## Advanced ORC properties

Usually, you do not need to modify ORC properties, but occasionally, Hortonworks Support advises making such changes.

| Key | Default Setting | Notes |
|---|---|---|
| orc.compress | ZLIB | Compression type (NONE, ZLIB, SNAPPY). |
| orc.compress.size | 262,144 | Number of bytes in each compression block. |
| orc.stripe.size | 268,435,456 | Number of bytes in each stripe. |
| orc.row.index.stride | 10,000 | Number of rows between index entries (>= 1,000). |
| orc.create.index | true | Sets whether to create row indexes. |
| orc.bloom.filter.columns | -- | Comma-separated list of column names for which a Bloom filter must be created. |
| orc.bloom.filter.fpp | 0.05 | False positive probability for a Bloom filter. Must be greater than 0.0 and less than 1.0. |

# Improving performance using partitions

You can use partitions to significantly improve performance. You can design Hive table and materialized views partitions to map to physical directories on the file system. For example, a table partitioned by date-time can organize data loaded into Hive each day.

Using partitions can significantly improve performance. You can design Hive table and materialized views partitions to map to physical directories on the file system. For example, a table partitioned by date-time can organize data loaded into Hive each day.

Large deployments can have tens of thousands of partitions. Partition pruning occurs indirectly when Hive discovers the partition key during query processing. For example, after joining with a dimension table, the partition key

might come from the dimension table. A query filters columns by partition, pruning partition scanning to one or a few matching partitions. Partition pruning occurs directly when a partition key is present in the WHERE clause. Partitioned columns are virtual, not written into the main table because these columns are the same for the entire partition. In a SQL query, you define the partition as shown in the following example:

```
CREATE TABLE sale(id in, amount decimal)
PARTITIONED BY (xdate string, state string);
```

To insert data into this table, you specify the partition key for fast loading:

```
INSERT INTO sale (xdate='2016-03-08', state='CA')
SELECT * FROM staging_table
WHERE xdate='2016-03-08' AND state='CA';
```

You do not need to specify dynamic partition columns. Hive generates a partition specification if you enable dynamic partitions.

**hive-site.xml settings for loading 1 to 9 partitions:**

```
SET
 hive.exec.dynamic.partition.mode=nonstrict;
SET
 hive.exec.dynamic.partition=true;
```

For bulk-loading data into partitioned ORC tables, you use the following property, which optimizes the performance of data loading into 10 or more partitions.

**hive-site.xml setting for loading 10 or more partitions:**

```
hive.optimize.sort.dynamic.partition=true
```

### Examples of a query on partitioned data

```
INSERT INTO sale (xdate, state)
SELECT * FROM staging_table;
```

Follow these best practices when you partition tables and query partitioned tables:

- Never partition on a unique ID.
- Size partitions so that on average they are greater than or equal to 1 GB.
- Formulate a query so that it does not process more than 1000 partitions.

# Handling bucketed tables

If you migrated data from earlier Apache Hive versions to Hive 3, you might need to handle bucketed tables that impact performance.

You can divide tables or partitions into buckets, which are stored in the following ways:

- As files in the directory for the table.
- As directories of partitions if the table is partitioned.

Using buckets in new Hive 3 tables is not necessary.

A common challenge related to using buckets is maintaining query performance while the workload or data scales up or down. For example, you could have an environment that operates smoothly using 16 buckets to support 1000 users, but a spike in the number of users to 100,000 for a day or two creates problems if you do not promptly tune the buckets and partitions. Tuning the buckets is complicated by the fact that after you have constructed a table with buckets, the entire table containing the bucketed data must be reloaded to reduce, add, or eliminate buckets.

With Tez, you only need to deal with the buckets of the biggest table. If workload demands change rapidly, the buckets of the smaller tables dynamically change to complete table JOINs.

You perform the following tasks related to buckets:

- Setting hive-site.xml to enable buckets

  SET hive.tez.bucket.pruning=true
- Bulk-loading tables that are both partitioned and bucketed:

  When you load data into tables that are both partitioned and bucketed, set the following property to optimize the process:

  SET hive.optimize.sort.dynamic.partition=true

If you have 20 buckets on user_id data, the following query returns only the data associated with user_id = 1:
SELECT * FROM tab WHERE user_id = 1;

To best leverage the dynamic capability of table buckets on Tez, adopt the following practices:

- Use a single key for the buckets of the largest table.
- Usually, you need to bucket the main table by the biggest dimension table. For example, the sales table might be bucketed by customer and not by merchandise item or store. However, in this scenario, the sales table is sorted by item and store.
- Normally, do not bucket and sort on the same column.

A table that has more bucket files than the number of rows is an indication that you should reconsider how the table is bucketed.

# Improving performance using the cost-based optimizer

A cost-based optimizer (CBO) generates efficient query plans, but to effectively use the CBO to optimize Hive data, you must generate column statistics for tables.

The CBO, powered by Apache Calcite, is a core component in the Hive query processing engine. The CBO optimizes plans for executing a query, calculates the cost, and selects the least expensive plan to use. In addition to increasing the efficiency of execution plans, the CBO conserves resources.

Hive not only enables CBO, but it also gathers table-level statistics by default; however, Hive does not use the CBO until you generate column statistics for tables. Hive does not enable column statistics by default because these statistics can be expensive to compute.

### How the CBO works

After parsing a query, a process converts the query to a logical tree (Abstract Syntax Tree) that represents the operations to perform, such as reading a table or performing a JOIN. Calcite applies optimizations, such as query rewrite, JOIN re-ordering, JOIN elimination, and deriving implied predicates to the query to produce logically equivalent plans. Bushy plans provide maximum parallelism. Each logical plan is assigned a cost that is based on distinct, value-based heuristics.

The Calcite plan pruner selects the lowest-cost logical plan. Hive converts the chosen logical plan to a physical operator tree, optimizes the tree, and converts the tree to a Tez job for execution on the Hadoop cluster.

### Explain plans

You can generate explain plans by running the EXPLAIN query command. An explain plan shows you the execution plan of a query by revealing the operations that occur when you run the query. Having a better understanding of the plan, you might rewrite the query or change Tez configuration parameters.

# Set up the cost-based optimizer and statistics

You can use the cost-based optimizer (CBO) and statistics to generate efficient query execution plans that can improve performance. You must generate column statistics to make CBO functional.

### About this task

In this task, you enable and configure the cost-based optimizer (CBO) and configure Hive to gather column and table statistics for evaluating query performance. Column and table statistics are critical for estimating predicate selectivity and cost of the plan. Certain advanced rewrites require column statistics.

In this task, you check, and set the following properties in the hive-site.xml configuration file:

- hive.stats.autogather

  Controls collection of table-level statistics.
- hive.stats.fetch.column.stats

  Controls collection of column-level statistics.
- hive.compute.query.using.stats

  Instructs Hive to use statistics when generating query plans.

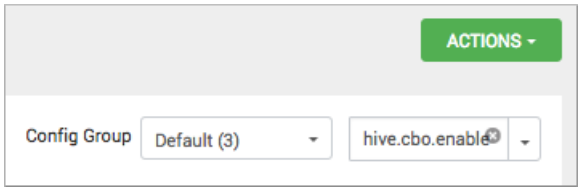
All of these properties are checked by default. You can manually generate the table-level statistics for newly created tables and table partitions using the ANALYZE TABLE statement.

### Before you begin

- You installed Ambari.
- You added the Apache Hive service and started all components.
- You have administrative privileges to configure Hive in Ambari.

### Procedure

1. In Ambari, select **Services** > **Hive** > **Configs**.
2. Enable cost-based optimization if you changed the default: In Filter, enter hive.cbo.enable, and check the checkbox.




3. Configure automatic gathering of table-level statistics for newly created tables and table partitions if you changed the default: In Filter, enter hive.stats.autogather, and check the checkbox.
4. Configure Hive to use statistics when generating query plans: In Filter, enter hive.compute.query.using.stats, and check the checkbox.
5. Restart Hive and any other affected services.

# Generate and view Apache Hive statistics

You can use statistics to optimize queries for improved performance. The cost-based optimizer (CBO) also uses statistics to compare query plans and choose the best one. By viewing statistics instead of running a query, you can sometimes get answers to your data questions faster.

### About this task

This task shows how to generate different types of statistics about a table.

### Procedure

1. Launch a hive shell and log in.
2. Gather statistics for the non-partitioned table mytable:
   ANALYZE TABLE mytable COMPUTE STATISTICS;
3. Confirm that the hive.stats.autogather property is enabled.
   a) In Ambari, select **Services** > **Hive** > **Configs**.
   b) In Filter, enter hive.stats.autogather.
4. View table statistics you generated:
   DESCRIBE EXTENDED mytable;
5. Gather column statistics for the table:

```
ANALYZE TABLE mytable COMPUTE STATISTICS FOR COLUMNS;
```

6. View column statistics for the name column in my_table in the my_db database:
   DESCRIBE FORMATTED my_db.my_table name;

### Related Information

Apache Hive Wiki language reference
Apache Hive Wiki - Statistics in Hive

## Statistics generation and viewing commands

You can manually generate table and column statistics, and then view statistics by issuing Hive queries. By default, Hive generates table statistics, but not column statistics, which you must generate manually to make cost-based optimization (CBO) functional.

### Commands for generating statistics

The following ANALYZE TABLE command generates statistics for tables and columns:

| | |
|---|---|
| **ANALYZE TABLE [table_name] COMPUTE STATISTICS;** | Gathers table statistics for non-partitioned tables. |
| **ANALYZE TABLE [table_name] PARTITION(partition_column) COMPUTE STATISTICS;** | Gathers table statistics for partitioned tables. |
| **ANALYZE TABLE [table_name] COMPUTE STATISTICS for COLUMNS [comma_separated_column_list];** | Gathers column statistics for the entire table. |
| **ANALYZE TABLE partition2 (col1="x") COMPUTE STATISTICS for COLUMNS;** | Gathers statistics for the partition2 column on a table partitioned on col1 with key x. |

### Commands for viewing statistics

You can use the following commands to view table and column statistics:

| | |
|---|---|
| **DESCRIBE [EXTENDED] table_name;** | View table statistics. The EXTENDED keyword can be used only if the hive.stats.autogather property is enabled in the hive-site.xml configuration file. |

| | |
|---|---|
| **DESCRIBE FORMATTED [db_name.]table_name [column_name] [PARTITION (partition_spec)];** | View column statistics. |

## Optimization and planning properties

The Ambari wizard automatically tunes the optimization- and planner-related configuration properties of Hive, Tez, and YARN, and you rarely need to make changes. However, in the event you want to tune these properties, guidelines are available.

| Property | Setting Guideline If Manual Configuration Is Needed | Default Value in Ambari |
|---|---|---|
| hive.auto.convert.join.<br>noconditionaltask.size | one-third of -Xmx value | Auto-tuned: Depends on environment |
| hive.tez.container.size | Production Systems: 4 to 8 GB<br>Small VMs: 1 to 2 GB | Auto-tuned: Depends on environment |
| hive.tez.java.opts | -Xmx value must be 80% to 90% of container size | Auto-tuned: Depends on environment |
| tez.grouping.min.size | Decrease for better latency<br>Increase for more throughput | 16777216 |
| tez.grouping.max.size | Decrease for better latency<br>Increase for more throughput | 1073741824 |
| tez.grouping.split-waves | Increase to launch more containers<br>Decrease to enhance multitenancy | 1.7 |
| yarn.scheduler.minimum-allocation-mb | 1 GB is usually sufficient | Auto-tuned: Depends on environment |