

# Apache Hive Performance Tuning

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



# Contents

<b>Optimizing an Apache Hive data warehouse.....</b>	<b>3</b>
Key components of Hive warehouse processing.....	3
Preparations for tuning performance.....	5
Setting up LLAP.....	5
Enable YARN preemption.....	5
Enable interactive query.....	6
Configure an llap queue.....	7
Configure a custom LLAP queue.....	8
Configure other LLAP properties.....	10
Configure the HiveServer heap size.....	11
Configure Hive MetaStore event authorization.....	12
Save LLAP settings and restart services.....	12
Run an LLAP query.....	13
Configure a queue for batch processing.....	14
Set up multiple HiveServer instances.....	16
Connect a JDBC client to LLAP.....	17
<b>Use HiveServer Interactive UI.....</b>	<b>18</b>
<b>Query result cache and metastore cache.....</b>	<b>19</b>
<b>Tez execution engine properties.....</b>	<b>20</b>
<b>Monitoring Apache Hive performance.....</b>	<b>20</b>
<b>Monitoring LLAP resources.....</b>	<b>21</b>
<b>Maximizing storage resources using ORC.....</b>	<b>22</b>
Advanced ORC properties.....	23
<b>Improving performance using partitions.....</b>	<b>24</b>
<b>Handling bucketed tables.....</b>	<b>25</b>
<b>Improving performance using the cost-based optimizer.....</b>	<b>25</b>
Set up the cost-based optimizer and statistics.....	26
Generate and view Apache Hive statistics.....	27
Statistics generation and viewing commands.....	27
Optimization and planning properties.....	28

## 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 Query Language (HiveQL) 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 gauged 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.

### 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 API driver, such as JDBC or ODBC. 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). Consequently, the number of Apache Tez Application Masters 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.

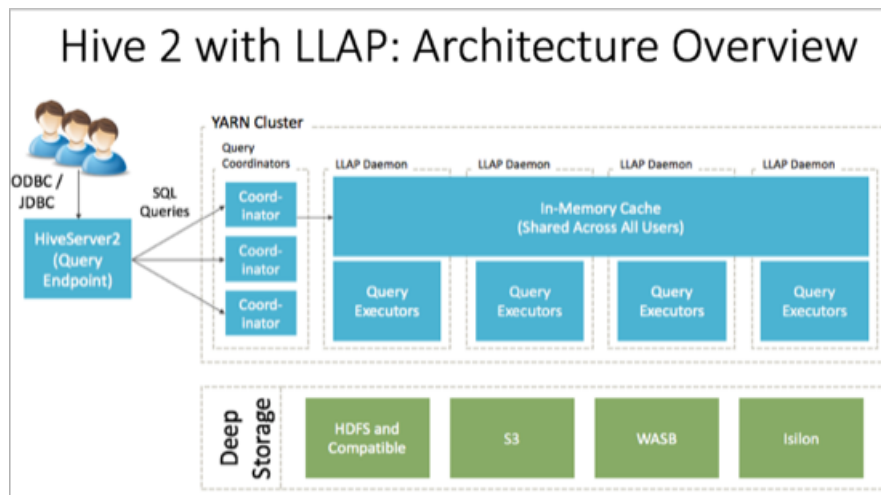
Hive on Tez is an advancement over earlier application frameworks for Hadoop data processing, such as using Hive on MapReduce, which is not supported in HDP 3.0 and later. The Tez framework is suitable for high-performance batch workloads.

After query compilation, HiveServer generates a Tez graph that is submitted to YARN. A Tez Application Master (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.

## 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 3.0 and later, 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.
- Activate and configure LLAP if you want 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 2.7.1 or later.
- You added and started YARN, Tez, and dependencies that Ambari prompts you to add.
- You added, started, and can run queries from the Hive service.

### About this task

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

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

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

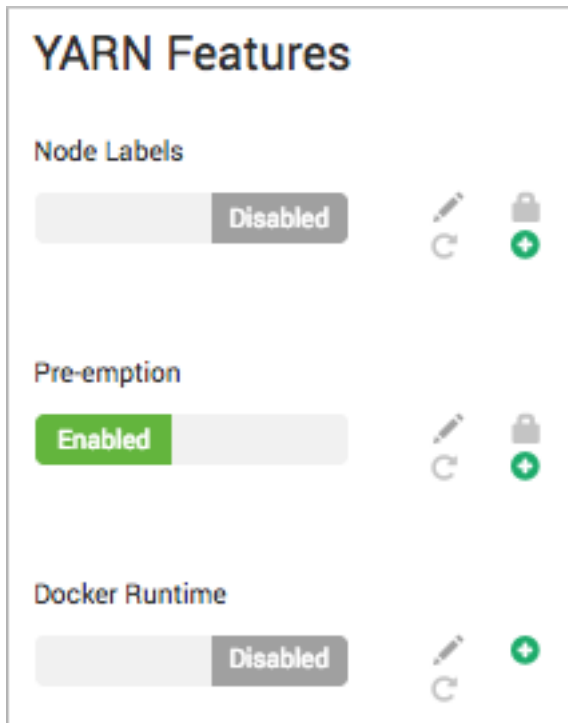
## Enable YARN preemption

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.

### Procedure

1. In Ambari, select **Services > YARN > Configs**.

- In YARN Features, set Pre-emption to Enabled (the default).



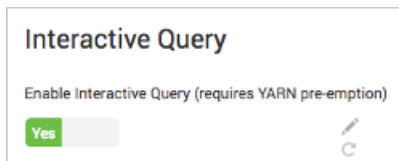
- Save the settings.

## Enable interactive query

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

### Procedure

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



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



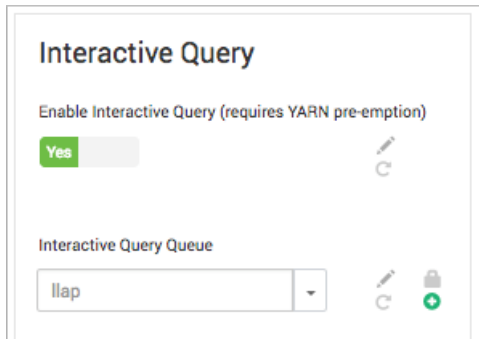
choice.

## Configure an llap queue

You set up an interactive query queue named llap by pointing the Hive service to a YARN queue, which you configure using the YARN Queue Manager.

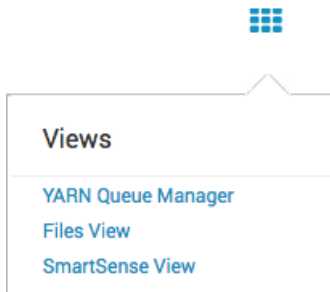
### Procedure

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

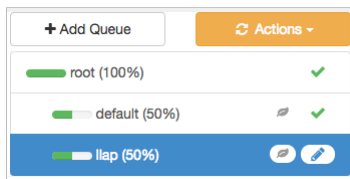


Depending on your YARN Capacity Scheduler settings, a queue named llap might or might not appear. This setting dedicates all the LLAP daemons and all the YARN Application Masters of the system to the single, specified queue.

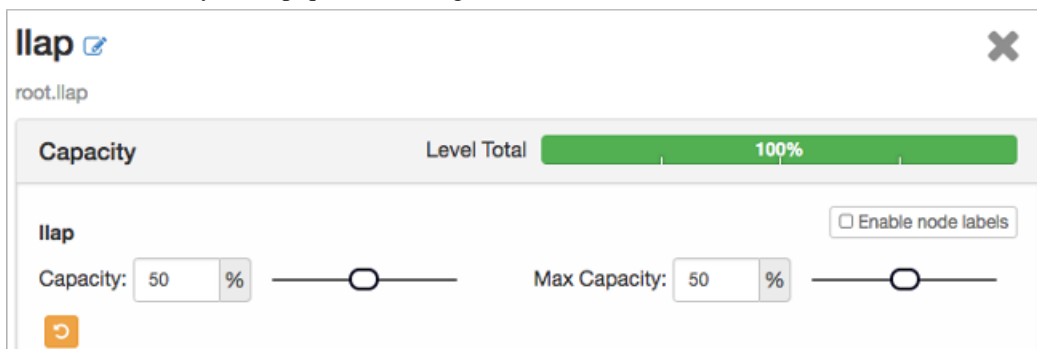
3. In Ambari, from Views, select **Services > YARN > YARN Queue Manager** from the hamburger menu icon.



4. If an llap queue does not exist, add a queue named llap. Otherwise, proceed to the next step.



5. If the llap queue is stopped, change the state to running.
6. Check that llap queue capacity and max capacity is not zero. Accept the percentage of total cluster capacity allocated towards your llap queue or change the values.



7. Select the llap queue under Add Queue.
8. In the Resources pane that appears on the right, set the following values: User Limit Factor to 1; Priority to greater than 0 (set 1 for example).
9. Select **Actions** > **Save** and Refresh Queues.
10. Restart any services as required.

## 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	<a href="/main/view/CAPACITY-SCHEDULER/auto_cs_instance">/main/view/CAPACITY-SCHEDULER/auto_cs_instance</a>	CAPACITY-SCHEDULER {1.0.0}
AUTO_FILES_INSTANCE	<a href="/main/view/FILES/auto_files_instance">/main/view/FILES/auto_files_instance</a>	FILES {1.0.0}
SMARTSENSE_AUTO_INSTANCE	<a href="/main/view/SMARTSENSE/smartsense_auto_instance">/main/view/SMARTSENSE/smartsense_auto_instance</a>	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.



The screenshot shows the YARN Queue Manager interface. At the top, there is a home icon and the text "/ YARN Queue Manager". Below this, there are two buttons: "+ Add Queue" and "Actions". Underneath, there are two queue entries: "root (100%)" and "default (100%)", each with a green progress bar and a checkmark. Below the queues is a "Scheduler" section with a green checkmark. The Scheduler configuration includes:
 

- Maximum Applications: 10000
- Maximum AM Resource: 20 %
- Node Locality Delay: 40
- Calculator: Default Resource Calculator
- Queue Mappings: (empty text box)
- Queue Mappings Override:  Disabled

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

The screenshot shows the YARN Queue Manager interface with the "Add Queue" dialog box open. The dialog box has a title bar with a home icon and "/ YARN Queue Manager". Below the title bar, there is a text input field labeled "Enter queue path..." with a red 'x' and a green checkmark button. Below the input field, there are two queue entries: "root (100%)" and "default (100%)", each with a green progress bar and a checkmark.

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:

The screenshot shows the "Interactive Query" settings page. The title is "Interactive Query". Below the title, there is a subtitle "Enable Interactive Query (requires YARN pre-emption)". Below the subtitle, there is a "Yes" button and a pencil icon.

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.

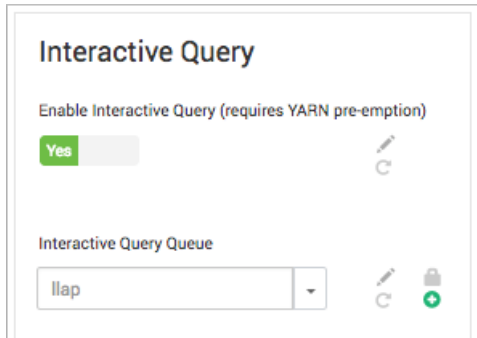
The screenshot shows the "Select HiveServer2 Interactive host" page. The title is "Select HiveServer2 Interactive host". Below the title, there is a subtitle "Assign HiveServer2 Interactive to host you want to run it on." and a note "If not present HDFS Client, MapReduce2 Client, Tez Client and YARN Client will also be installed on the selected host." Below the subtitle, there are three dropdown menus:
 

- NameNode: c7401.ambari.apache.org (3.7 GB, 1...)
- SNameNode: c7402.ambari.apache.org (3.7 GB, 1...)
- YARN Registry DNS: c7401.ambari.apache.org (3.7 GB, 1...)

 To the right of these dropdowns, there is a list of services for the selected host:
 

- NameNode
- YARN Registry DNS
- Timeline Service V2.0 Reader
- ResourceManager
- HiveServer2 Interactive (highlighted in green)
- ZooKeeper Server
- Grafana

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



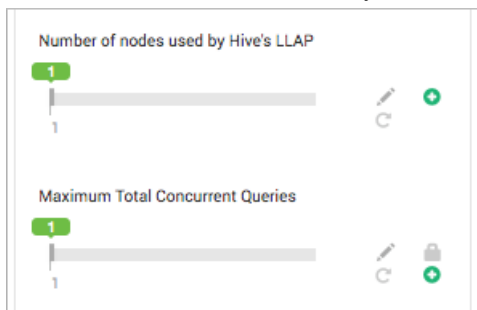
This action dedicates all the LLAP daemons and YARN ApplicationMasters of the system to the single, specified queue.

### Configure other LLAP properties

You can set up low-latency analytical processing (LLAP) using Ambari wizards that recommend properties for interactive querying in your environment.

#### Procedure

1. Set the Number of Nodes Used By Hive LLAP. For example, set 1 node.



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

2. Accept the Maximum Total Concurrent Queries or make changes.
3. Make changes to other settings to suit your environment, or accept the defaults as shown in the following example.

The screenshot shows the Ambari configuration interface for YARN settings. It contains four sections:

- Memory per Daemon:** A slider set to 1024MB. The scale ranges from 1024 MB to 23.434 GB, with a major tick at 12.217 GB. A green callout box highlights the current value of 1024MB.
- In-Memory Cache per Daemon:** A slider set to 0MB. The scale ranges from 0 MB to 23.434 GB, with a major tick at 11.717 GB. A green callout box highlights the current value of 0MB.
- Number of executors per LLAP Daemon:** A slider set to 6. The scale ranges from 1 to 7, with a major tick at 4. A green callout box highlights the current value of 6.
- Turn SSD Cache On?:** A toggle switch currently set to "No".

Each slider has a pencil icon for editing, a refresh icon, and a lock icon with a plus sign to indicate it can be unlocked.

4. In Ambari, select **Services > YARN > Configs > Advanced**.

5. 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.

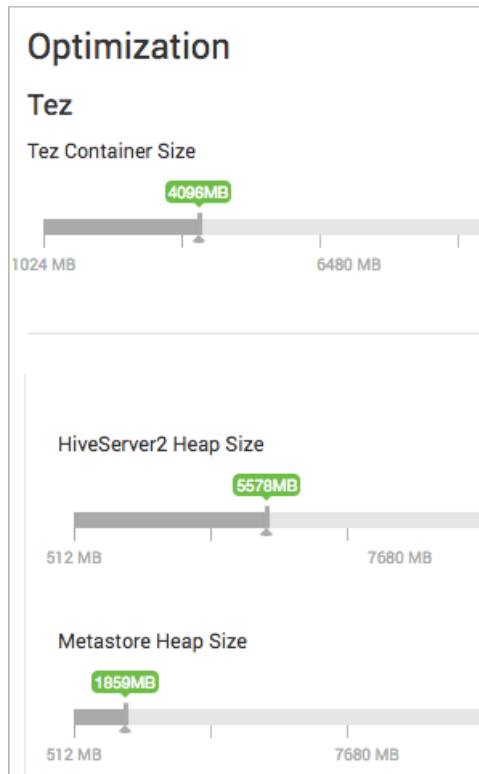
For example, if your cluster has 20 nodes, then divide 1 by 20 and enter 0.05 as the value of this property setting.

### 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.

### Configure Hive MetaStore event authorization

The default `hive.metastore.event.db.notification.api.auth` property configuration can prevent you from connecting to HiveServer Interactive process and using LLAP. You can change the property setting in `hive-site.xml` through Ambari.

### Procedure

1. In Ambari, in **Services > Hive > Configs > Advanced > Custom hive-site**, add the following property key:  
`hive.metastore.event.db.notification.api.auth`
2. Set the value of the property to false.
3. Save the change, and restart Hive components as required.

### 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.

**Summary**

Components

- Started HIVE METASTORE
- Started HIVESERVER2 INTERACTIVE
- Started HIVESERVER2
- Started MYSQL SERVER

3 Installed HIVE CLIENTS

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

**Quick Links**

- [Hive Dashboard \(Grafana\)](#)
- [HiveServer2 Interactive UI](#)
- [Jdbc Standalone Jar Download](#)

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

Home Local logs Metrics Dump Hive Configuration Stack Trace Llap Daemons

App Type	yarn-service	
App Id	application_1550171072120_0028	
State	RUNNING_ALL	
App Start Time	0	
Running Threshold Achieved	false	
Desired Instances	2	
Live Instances	2	
Launching Instances	0	
Running Instances	Container Id	Web Url
	container_e23_1550171072120_0028_01_000003	http://main-6.1.1.1:15002
	container_e23_1550171072120_0028_01_000002	http://main-7.1.1.1:15002

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

SUMMARY CONFIGS ALERTS 0 VERSIONS

Components + ADD

Status	Name	Type	Action
✓	HiveServer2 Interactive / Hive	Master	...
✓	DataNode / HDFS	Slave	
✓	HST Agent / SmartSense	Slave	
✓	Metrics Monitor / Ambari Metrics	Slave	

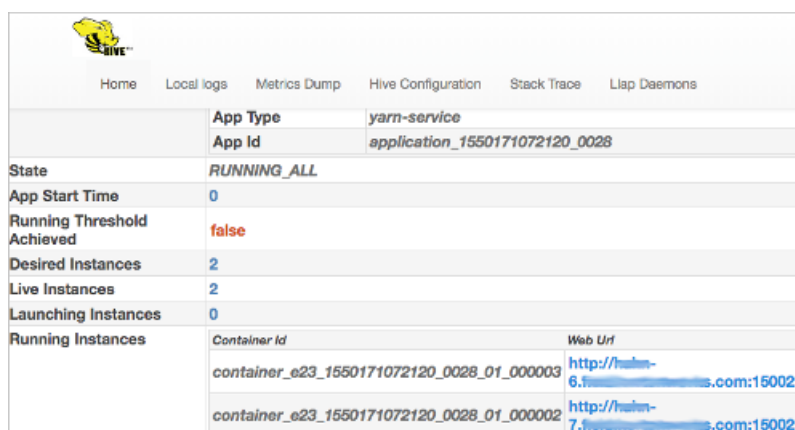
Restart  
Stop  
Turn On Maintenance Mode  
Delete  
Restart LLAP ∞

## Run an LLAP query

You connect to HiveServer through Beeline to run LLAP queries. In the connection string, you specify the FQDN of the node that runs HiveServer Interactive.

## Before you begin

- You setup LLAP and restarted services.
- The HiveServer Interactive UI, which you access from **Summary > Quick Links > HiveServer Interactive UI** indicates that LLAP is running.



Hive							
<a href="#">Home</a> <a href="#">Local logs</a> <a href="#">Metrics Dump</a> <a href="#">Hive Configuration</a> <a href="#">Stack Trace</a> <a href="#">Liap Daemons</a>							
<b>App Type</b>	yarn-service						
<b>App Id</b>	application_1550171072120_0028						
<b>State</b>	RUNNING_ALL						
<b>App Start Time</b>	0						
<b>Running Threshold Achieved</b>	false						
<b>Desired Instances</b>	2						
<b>Live Instances</b>	2						
<b>Launching Instances</b>	0						
<b>Running Instances</b>	<table border="1"> <thead> <tr> <th>Container Id</th> <th>Web Url</th> </tr> </thead> <tbody> <tr> <td>container_e23_1550171072120_0028_01_000003</td> <td>http://hdm-6.1550171072120.com:15002</td> </tr> <tr> <td>container_e23_1550171072120_0028_01_000002</td> <td>http://hdm-7.1550171072120.com:15002</td> </tr> </tbody> </table>	Container Id	Web Url	container_e23_1550171072120_0028_01_000003	http://hdm-6.1550171072120.com:15002	container_e23_1550171072120_0028_01_000002	http://hdm-7.1550171072120.com:15002
Container Id	Web Url						
container_e23_1550171072120_0028_01_000003	http://hdm-6.1550171072120.com:15002						
container_e23_1550171072120_0028_01_000002	http://hdm-7.1550171072120.com:15002						

## Procedure

- On the command-line of a node in the cluster, enter the command, substituting the FQDN of your HiveServer Interactive node, to connect to HiveServer Interactive on port 10500.

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

- 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 than your non-LLAP queries.

## 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

- 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
```

**Scheduler**

---

Capacity Scheduler

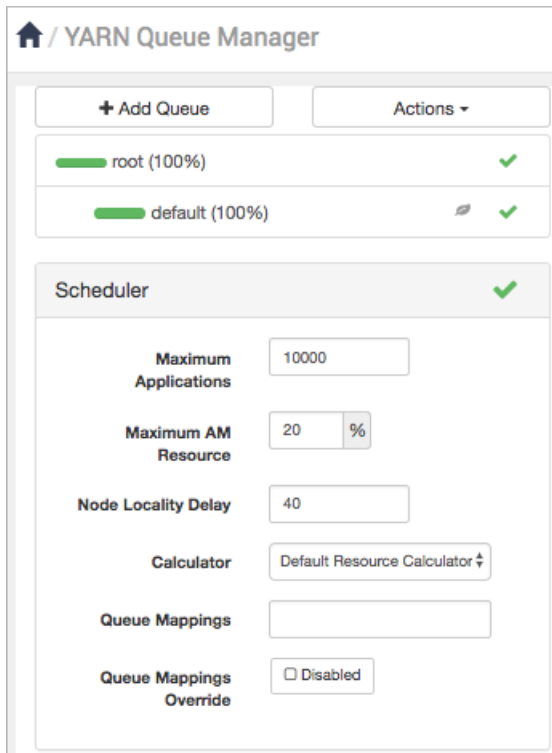
```
capacity-scheduler=null
yarn.scheduler.capacity.default.minimum-user-limit-percent=100
yarn.scheduler.capacity.maximum-am-resource-percent=0.2
yarn.scheduler.capacity.maximum-applications=10000
yarn.scheduler.capacity.node-locality-delay=40
yarn.scheduler.capacity.resource-calculator=org.apache.hadoop.yarn.
yarn.scheduler.capacity.root.accessible-node-labels=*
yarn.scheduler.capacity.root.acl_administer_queue=*
yarn.scheduler.capacity.root.acl_submit_applications=*
yarn.scheduler.capacity.root.capacity=100
yarn.scheduler.capacity.root.default.acl_administer_jobs=*
yarn.scheduler.capacity.root.default.acl_submit_applications=*
yarn.scheduler.capacity.root.default.capacity=100
yarn.scheduler.capacity.root.default.maximum-capacity=100
yarn.scheduler.capacity.root.default.state=RUNNING
yarn.scheduler.capacity.root.default.user-limit-factor=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.

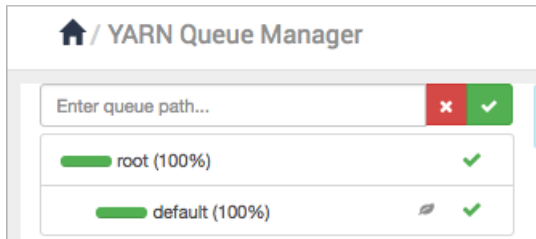
4. From the Ambari dashboard, select **ADMIN > Manage Ambari > Views**.

Name	URL	View Type
AUTO_CS_INSTANCE	<a href="/main/view/CAPACITY-SCHEDULER/auto_cs_instance">/main/view/CAPACITY-SCHEDULER/auto_cs_instance</a>	CAPACITY-SCHEDULER {1.0.0}
AUTO_FILES_INSTANCE	<a href="/main/view/FILES/auto_files_instance">/main/view/FILES/auto_files_instance</a>	FILES {1.0.0}
SMARTSENSE_AUTO_INSTANCE	<a href="/main/view/SMARTSENSE/smartsense_auto_instance">/main/view/SMARTSENSE/smartsense_auto_instance</a>	SMARTSENSE {1.5.0.2.7.0.0-703}

5. Click the URL for the view named AUTO\_CS\_INSTANCE, which is the capacity scheduler view.

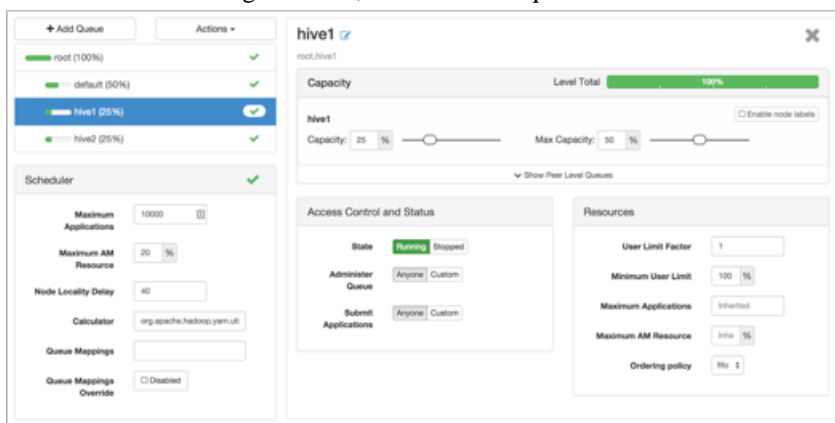


6. In the YARN Queue Manager, click Add Queue.



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:



## Set up multiple HiveServer instances

If your cluster needs to accommodate more than 40 concurrent executing queries, you should create a new HiveServer instance. You can set up HiveServer instances, which can share the same metastore database for different purposes,



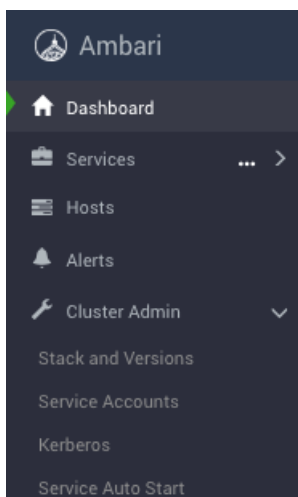
such as ETL, interactive queries, running multiple applications using different settings, or load-balancing using ZooKeeper.

### Before you begin

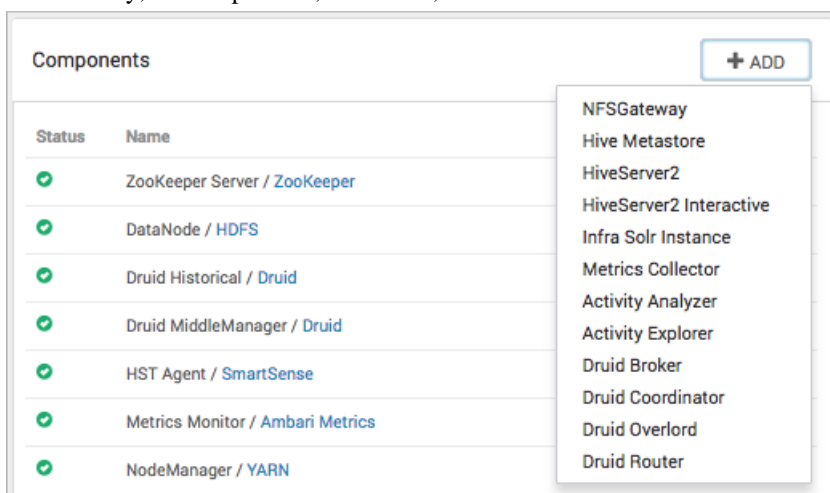
- You installed Hive using Ambari and Hive is running.

### Procedure

- In Ambari, select Hosts.



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



## Connect a JDBC client to LLAP

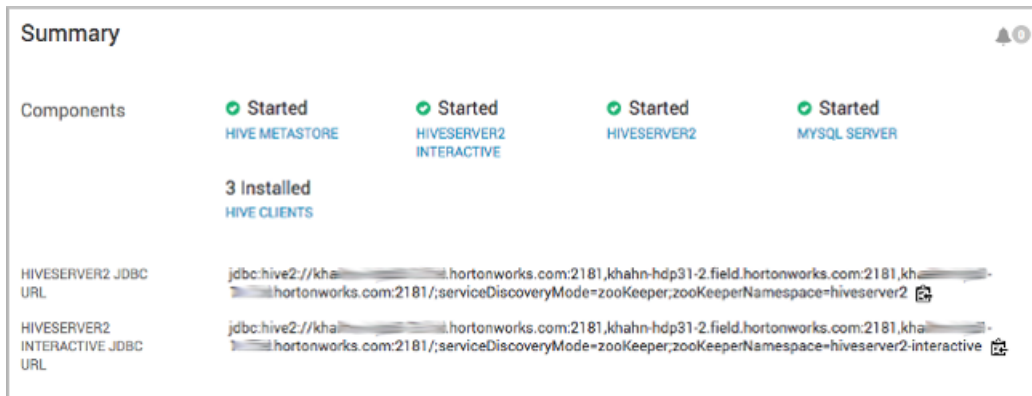
As administrator, you can provide users with the JDBC URL generated by HiveServer, so they can issue queries to the Hive LLAP instance.

### 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.

## 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

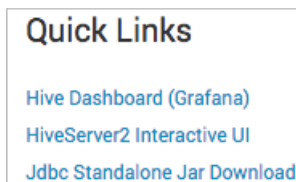
- You installed Hive using Ambari and Hive is running.
- You enabled LLAP in the cluster.

### About this task

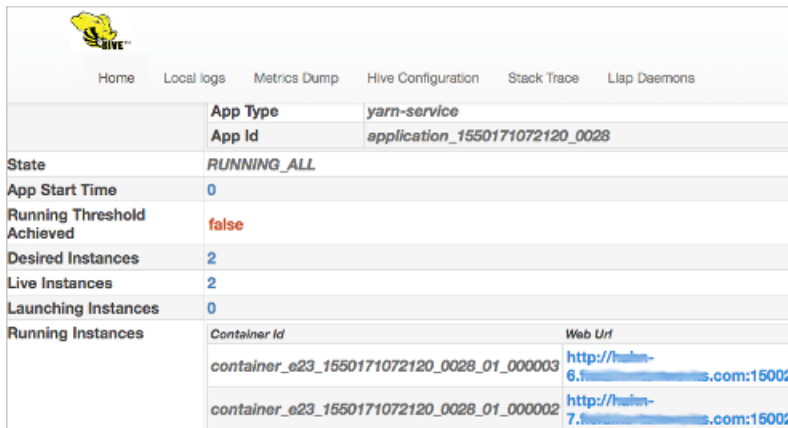
In this task you open the HiveServer Interactive UI and view executing queries in real time or a recent history of queries. From the UI, you can also access running LLAP daemons.

### 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).



Hive		
<a href="#">Home</a> <a href="#">Local logs</a> <a href="#">Metrics Dump</a> <a href="#">Hive Configuration</a> <a href="#">Stack Trace</a> <a href="#">Llap Daemons</a>		
<b>App Type</b>	yarn-service	
<b>App Id</b>	application_1550171072120_0028	
<b>State</b>	RUNNING_ALL	
<b>App Start Time</b>	0	
<b>Running Threshold Achieved</b>	false	
<b>Desired Instances</b>	2	
<b>Live Instances</b>	2	
<b>Launching Instances</b>	0	
<b>Running Instances</b>	<b>Container Id</b>	<b>Web Url</b>
	container_e23_1550171072120_0028_01_000003	http://train-6.1550171072120.com:15002
	container_e23_1550171072120_0028_01_000002	http://train-7.1550171072120.com:15002

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

## 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 perform 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 access 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

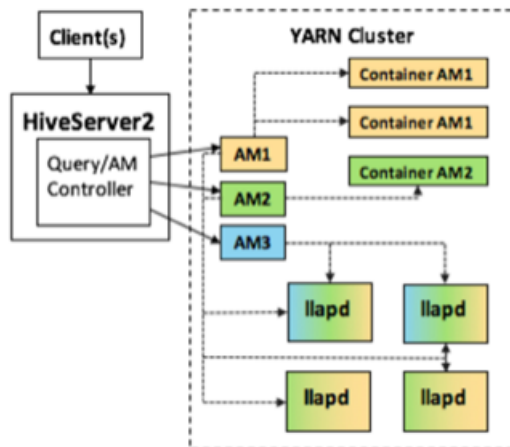
Metrics that show the operating status for a specific Hive LLAP Daemon.

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

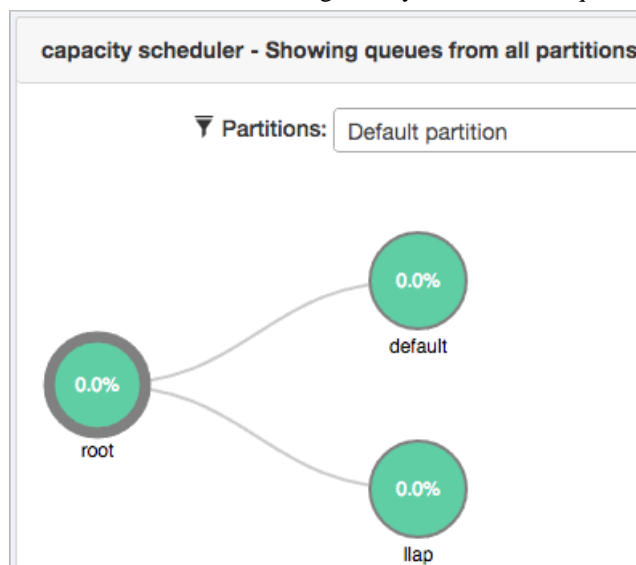
As administrator, you can monitor a cluster to conserve resources when running low-latency analytical processing (LLAP). You can monitor the queries in real-time using the YARN ResourceManager Web UI, Slider, or YARN command-line tools. Running through Slider, you can open the cluster, share resources with other applications, remove the cluster, and manage resources.

After setup, Hive LLAP is transparent to Apache Hive users and business intelligence tools. Interactive queries run on Apache Hadoop YARN as an Apache Slider application. 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:

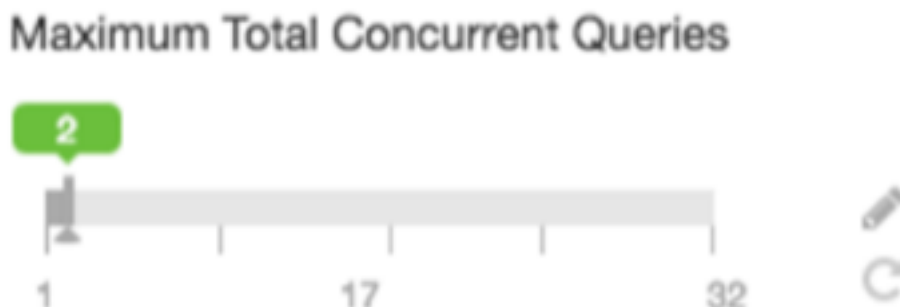


On your cluster, an extra HiveServer instance is installed that is dedicated to interactive queries. You can see this HiveServer instance listed in the Hive Summary page of Ambari.

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:

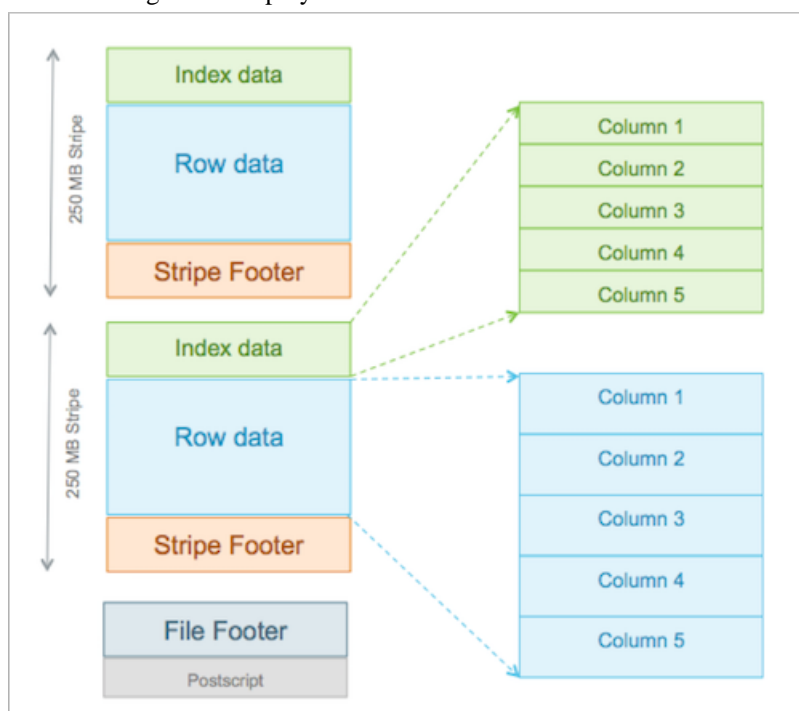


## Maximizing storage resources using ORC

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 ( $\geq 1,000$ ).
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 int, 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 HiveQL 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.

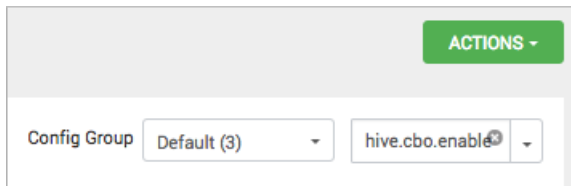
All of these properties are checked by default. You can manually generate the table-level statistics for newly created tables and table partitions using the ANALYZE TABLE statement.

### 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.

## 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 students table:  
`ANALYZE TABLE mytable COMPUTE STATISTICS;`
3. Enable `hive.stats.autogather` property required for the `DESCRIBE EXTENDED` command.
  - a) In Ambari, select **Services** > **Hive** > **Configs**.
  - b) In Filter, enter `hive.stats.autogather`, and check the checkbox.
4. View table statistics you generated:  
`DESCRIBE EXTENDED mytable;`
5. 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.
```

<b>ANALYZE TABLE</b> [table_name] <b>PARTITION</b> (partition_column) <b>COMPUTE STATISTICS</b> ;	Gathers table statistics for partitioned tables.
<b>ANALYZE TABLE</b> [table_name] <b>COMPUTE STATISTICS</b> for COLUMNS [comma_separated_column_list];	Gathers column statistics for the entire table.
<b>ANALYZE TABLE</b> partition2 (col1="x") <b>COMPUTE STATISTICS</b> 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:

<b>DESCRIBE</b> [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.
<b>DESCRIBE FORMATTED</b> [db_name.]table_name.column_name;	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. noconditionaltask.size	one-third of -Xmx value	Auto-tuned: Depends on environment
hive.tez.container.size	Production Systems: 4 to 8 GB 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 Increase for more throughput	16777216
tez.grouping.max.size	Decrease for better latency Increase for more throughput	1073741824
tez.grouping.split-waves	Increase to launch more containers Decrease to enhance multitenancy	1.7
yarn.scheduler.minimum-allocation-mb	1 GB is usually sufficient	Auto-tuned: Depends on environment