Hortonworks Data Platform

Apache Hive Performance Tuning

(December 15, 2017)

docs.hortonworks.com

Hortonworks Data Platform: Apache Hive Performance Tuning

The Hortonworks Data Platform, powered by Apache Hadoop, is a massively scalable and 100% open source platform for storing, processing and analyzing large volumes of data. It is designed to deal with data from many sources and formats in a very quick, easy and cost-effective manner. The Hortonworks Data Platform consists of the essential set of Apache Hadoop projects including MapReduce, Hadoop Distributed File System (HDFS), HCatalog, Pig, Hive, HBase, ZooKeeper and Ambari. Hortonworks is the major contributor of code and patches to many of these projects. These projects have been integrated and tested as part of the Hortonworks Data Platform release process and installation and configuration tools have also been included.

Unlike other providers of platforms built using Apache Hadoop, Hortonworks contributes 100% of our code back to the Apache Software Foundation. The Hortonworks Data Platform is Apache-licensed and completely open source. We sell only expert technical support, training and partner-enablement services. All of our technology is, and will remain, free and open source.

Please visit the Hortonworks Data Platform page for more information on Hortonworks technology. For more information on Hortonworks services, please visit either the Support or Training page. Feel free to Contact Us directly to discuss your specific needs.

Table of Contents

List of Figures

List of Tables

1. Optimizing an Apache Hive Data Warehouse

Using a Hive-based data warehouse requires setting up the appropriate environment for your needs. After you establish the computing paradigm and architecture, you can tune the data warehouse infrastructure, interdependent components, and your client connection parameters to improve the performance and relevance of business intelligence (BI) and other data-analytic applications.

Tuning Hive and other Apache components that run in the background to support processing of HiveQL is particularly important as the scale of your workload and database volume increases. When your applications query data sets that constitute a large-scale enterprise data warehouse (EDW), tuning the environment and optimizing Hive queries are often part of an ongoing effort by IT or DevOps teams to ensure service-level agreement (SLA) benchmarks or other performance expectations.

Increasingly, most enterprises require that Hive queries run against the data warehouse with *low-latency analytical processing*, which is often referred to as *LLAP* by Hortonworks. LLAP of real-time data can be further enhanced by integrating the EDW with the Druid business intelligence engine.

Tip

The best approach is to use Apache Ambari to configure and monitor applications and queries that run on a Hive data warehouse. These tips are described throughout this guide.

1.1. Hive Processing Environments

The environment that you use to process queries and return results can depend on one or more factors, such as the capacity of your system resources, how in-depth you want to analyze data, how quickly you want queries to return results, or what tradeoffs that you can accept to favor one model over another.

1.1.1. Overall Architecture

A brief overview of the components and architecture of systems using Hive EDW for data processing is in the Hive Architectural Overview of HDP 2.5. With a few exceptions, the architecture information there applies to both batch processing and LLAP of Hive queries. However, there are some differences in the way the components of an environment processing batch workloads operate from the functioning of the same components in a Hive LLAP environment.

1.1.2. Dependencies for Optimal Hive Query Processing

Increasingly, enterprises want to run SQL workloads that return faster results than batch processing can provide. Hortonworks Data Platform (HDP) supports Hive LLAP, which enables application development and IT infrastructure to run queries that return real-time

or near-real-time results. Use cases for implementing this technology include environments where users of business intelligence (BI) tools or web dashboards need to accelerate analysis of data stored in a Hive EDW.

A performance benchmark that enterprises increasingly want to reach with data analytics applications is support for interactive queries. *Interactive queries* are queries on Hive data sets that meet low-latency benchmarks that are variably gauged but for Hive LLAP in HDP is specified as 15 seconds or less.

Important

Hive LLAP with Apache Tez utilizes newer technology available in Hive 2.x to be an increasingly needed alternative to other execution engines like MapReduce and earlier implementations of Hive on Tez. Tez runs in conjunction with Hive LLAP to form a newer execution engine architecture that can support faster queries.

Important

The Hive LLAP with Tez engine requires a different Apache Hadoop YARN configuration from the configuration required for environments where Hive on Tez is the execution engine. With Ambari 2.5.0 and later versions, you can more easily enable and configure YARN components that are the foundation of Hive LLAP than you could in previous HDP releases.

1.1.3. Connectivity to Admission Control (HiveServer2)

HiveServer2 is a service that enables multiple clients to simultaneously execute queries against Hive using an open API driver, such as JDBC or ODBC.

For optimal performance, use HiveServer2 as the connectivity service between your client application and the Hive EDW. HiveServer1 is deprecated because HiveServer2 has improvements for multiclient concurrency and authentication. Also, HiveServer2 is designed to provide better support for open API clients like JDBC and ODBC.

HiveServer2 is one of several architectural components for admission control, which enables optimal Hive performance when multiple user sessions generate asynchronous threads simultaneously. Admission control operates by scaling the Hive processing of concurrent queries to a workload that is suited to the system resources and to the total demand of incoming threads, while holding the other queries for later processing or cancelling the queries if conditions warrant this action. Admission control is akin to “connection pooling” in RDBMS databases.

To optimize Hive performance, you must set parameters that affect admission control according to your needs and system resources.

Important

HiveServer2 coordinates admission control in conjunction with YARN and Apache Tez for batch queries and with YARN and the LLAP daemons for interactive queries.

1.1.4. Execution Engines (Apache Tez and Hive LLAP)

Both the Hive on Tez engine for batch queries and the enhanced Tez + Hive LLAP engine run on YARN nodes.

1.1.4.1. Tez Execution on YARN

Hive on Tez is an advancement over earlier application frameworks for Hadoop data processing, such as using Hive on MapReduce2 or MapReduce1. The Tez framework is required for high-performance batch workloads. Tez is also part of the execution engine for Hive LLAP.

After query compilation, HiveServer2 generates a Tez graph that is submitted to YARN. A Tez ApplicationMaster (AM) monitors the query while it is running.

The maximum number of queries that can be run concurrently is limited by the number of ApplicationMasters.

1.1.4.2. Hive LLAP Execution Engine

The architecture of Hive LLAP is illustrated in the following

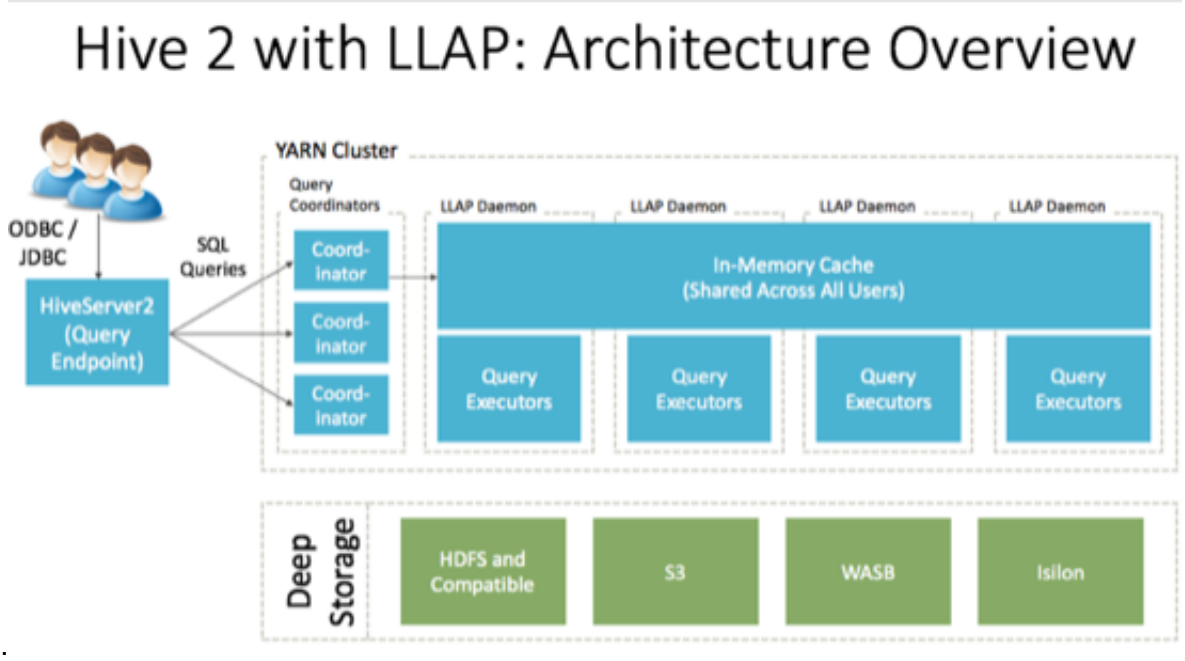


diagram.

- HiveServer2: provides JDBC and ODBC interface, and query compilation
- Query coordinators: coordinate the execution of a single query LLAP daemon: persistent server, 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: cache inside the LLAP daemon that is shared across all users

1.1.4.3. Workload Management with Queues and Containers (Hive, YARN, and Tez)

1.1.4.3.1. Batch Processing

Each queue must have the capacity to support one complete Tez Application, as defined by its ApplicationMaster (AM). Consequently, the maximum number of queries that can be run concurrently is also limited by the number of Apache Tez Application Masters.

A Hive-based analytic application relies on execution resources called *YARN containers*. Containers are defined by the Hive configuration. 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.

1.1.4.3.2. Interactive Workloads

Interactive workloads operate with YARN and queues differently from the way that batch workloads manage workloads.

When using the Hive LLAP on Tez engine, Admission Control is handled differently than for earlier Hive on Tez implementations. Resources are managed by Hive LLAP globally, rather than each Tez session managing its own.

Hive LLAP has its own resource scheduling and pre-emption built in that doesn't rely on YARN. As a result, a single queue is needed to manage all LLAP resources. In addition, each LLAP daemon runs as a single YARN container.

1.1.4.4. SQL Planner and Optimizer (Apache Hive and Apache Calcite)

A cost-based optimizer (CBO) generates more efficient query plans. In Hive, the CBO is enabled by default, but it requires that column statistics be generated for tables. Column statistics can be expensive to compute so they are not automated. Hive has a CBO that is based on Apache Calcite and an older physical optimizer. All of the optimizations are being migrated to the CBO. The physical optimizer performs better with statistics, but the CBO requires statistics.

1.1.4.5. Storage Formats

Hive supports various file formats. You can write your own SerDes (Serializers, Deserializers) interface to support new file formats.

Tip

The Optimized Row Columnar (ORC) file format for data storage is recommended because this format provides the best Hive performance overall.

1.1.4.6. Storage Layer (Example: HDFS Filesystem)

While a Hive EDW can run on one of a variety of storage layers, HDFS and Amazon S3 are among the most prevalently used and known filesystems for data analytics that run in the Hadoop stack. Amazon S3 is a commonly used filesystem used for a public cloud infrastructure.

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

Depending on your environment, you can tune the filesystem to optimize Hive performance by configuring compression format, stripe size, partitions, and buckets. Also, you can create bloom filters for columns frequently used in point lookups.

1.2. Setting up Hive LLAP

Important

Using Ambari 2.5.0+ to enable Hive LLAP and configure most of its basic parameters is highly recommended for most users. Ambari not only has a GUI to ease the tasks, but also contains multiple wizards that can automatically tune interactive query property settings to suit your environment.

While most of the Hive LLAP installation and configuration steps can be completed in Ambari, you must manually configure two properties in the `yarn-site.xml` file before sliding the **Enable Interactive Query** toggle to "Yes." Then there are two paths for enabling Hive LLAP using Ambari: Typical Setup and Advanced Setup. Typical Setup is recommended for most users because it requires less decision-making and leverages more autotuning features of Ambari than the Advanced Setup.

1.2.1. Enabling YARN Preemption for Hive LLAP

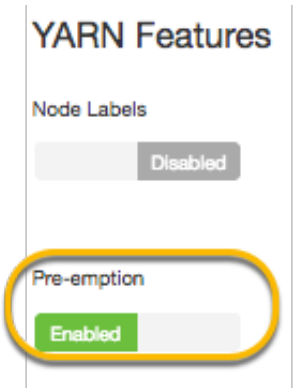
About this Task

You must enable and configure YARN preemption, which directs the Capacity Scheduler to position a Hive LLAP queue as the top-priority workload to run among cluster node resources. See YARN Preemption for more information about how YARN preemption functions.

Steps

1. In Ambari, select **Services** > **YARN** > **Configs** tab > **Settings** subtab.
2. Set the **Pre-emption** slider of the **YARN Features** section to Enabled:

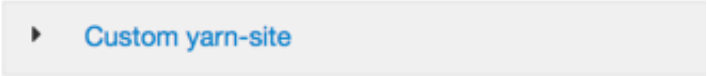
Figure 1.1. YARN Features Pane



3. Click the **Advanced** subtab.

4. Set the `yarn-site.xml` properties required to enable Hive LLAP.

   a. Open the **Custom yarn-site** drop-down menu.

   b. Use the **Add Property ...** link in the GUI to add and configure the properties as documented in the following table.

   Table 1.1. Manual Configuration of Custom yarn-site Properties for Enabling Hive LLAP

Property Name	Recommended Setting
yarn.resourcemanager.monitor. *capacity.preemption.natural_termination_factor*	**1**
yarn.resourcemanager.monitor.capacity. *preemption.total_preemption_per_round*	Calculate the value by dividing 1 by the number of cluster nodes. Enter the value as a decimal. *Example:* If your cluster has 20 nodes, then divide 1 by 20 and enter **0.05** as the value of this property setting.

5. Click **Save** in the upper right area of the window.

Next Step

Complete either the Enable Hive LLAP: Typical Setup task or the Enable Hive LLAP: Advanced Setup in Ambari in the following sections.

1.2.2. Enable Hive LLAP: Typical Setup

About this Task

Follow this procedure if you are new to Hive LLAP or prefer to let autotuning features of Ambari configure interactive queries.

Prerequisites

- Installation of Ambari 2.5.x
- The Hive Service and other interdependencies as prompted in Ambari must be running.
- YARN preemption must be enabled and configured as documented in the Enabling YARN Preemption for Hive LLAP section above.

Steps

1. Select the **Hive** service in the Ambari dashboard.

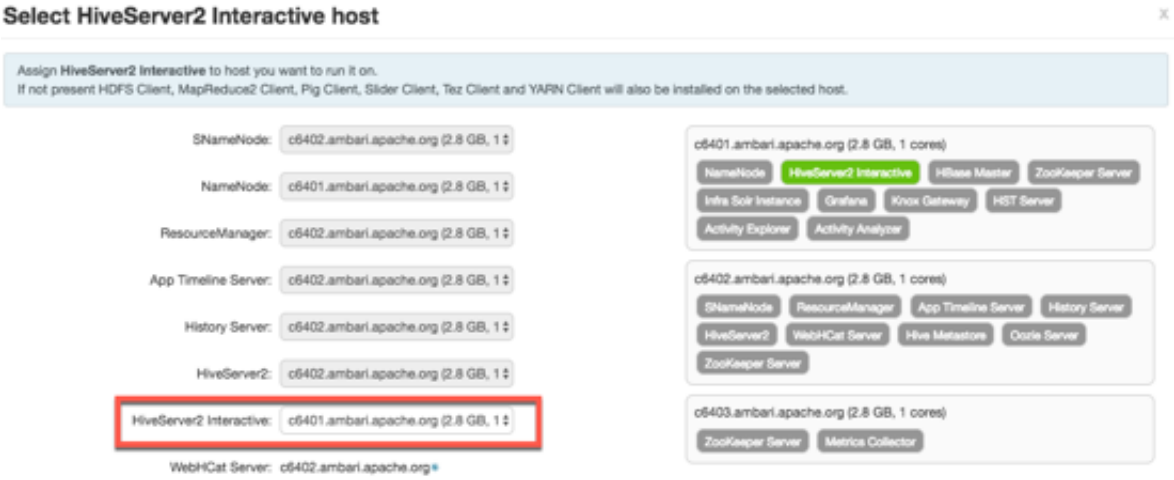
2. Click the **Configs** tab.

3. In the **Settings** tab, locate the **Interactive Query** section and set the **Enable Interactive Query** slider to **Yes**.

Figure 1.2. Enable Interactive Query Toggle on the Settings Tab

4. Select the server to host the HiveServer2 Interactive instance in the **HiveServer2 Interactive** field. In most cases, you can keep the default server host assignment.

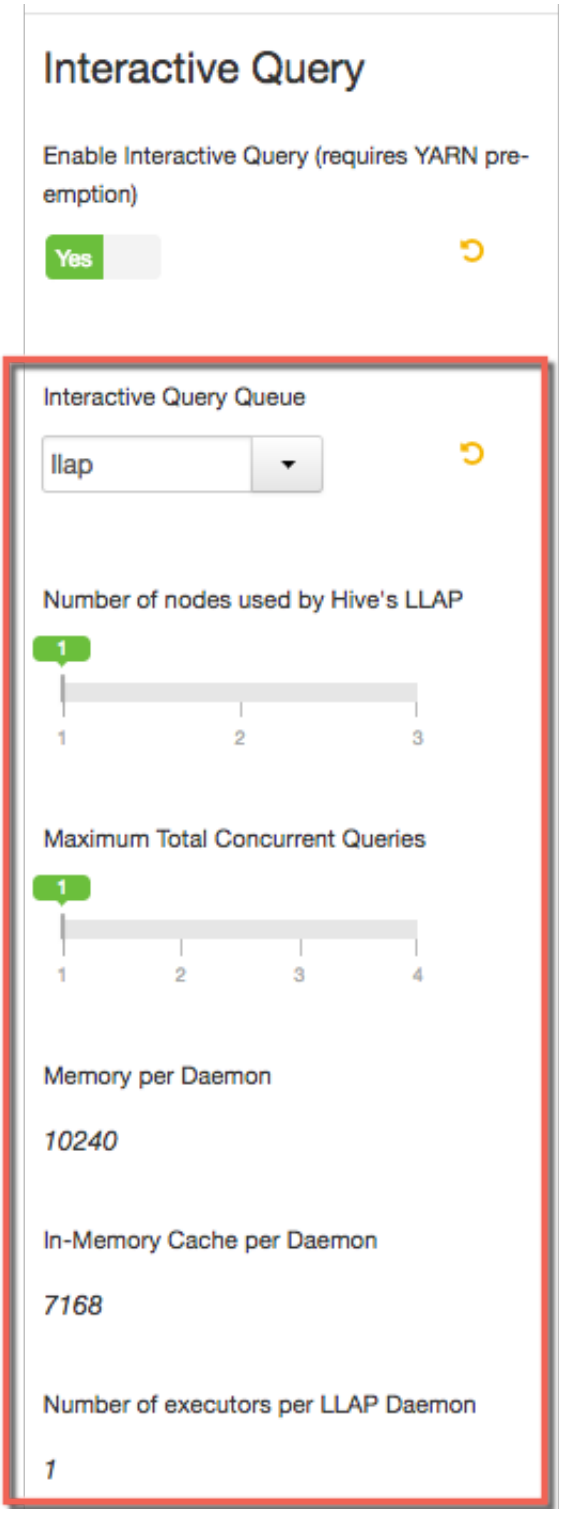
Figure 1.3. Select HiveServer2 Interactive Host Window



5. Click **Select** in the **Select HiverServer2 Interactive host** window.

6. When the **Settings** subtab opens again, review the additional configuration fields that appear in the **Interactive Query** section of the window:

Figure 1.4. Enabled Interactive Query Configuration Settings



Retain **llap** as the setting in the **Interactive Query Queue** drop-down menu. This setting dedicates all the LLAP daemons and all the YARN ApplicationMasters of the system to the single, specified queue.

7. Set the **Number of nodes used by Hive LLAP** slider to the number of cluster nodes on which to run Hive LLAP. LLAP automatically deploys to the nodes, and you do not need to label the nodes.

8. Set the **Maximum Total Concurrent Queries** slider to the maximum number of concurrent LLAP queries to run. The Ambari wizard calculates and displays a range of values in the slider based on the number of nodes that you selected and the number of CPUs in the Hive LLAP cluster.

9. Review the following settings, which are autogenerated for informational purposes only. (No interactive elements allow you to directly change the values.)

   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 executors per daemon: for example, the number of fragments that can execute in parallel on a daemon

10. Review the property settings outside the **Interactive Query** section of the window to learn how the Hive LLAP instance is configured. The Ambari wizard calculates appropriate values for most other properties on the **Settings** subtab, based on the configurations in the **Interactive Query** section of the window.

Important

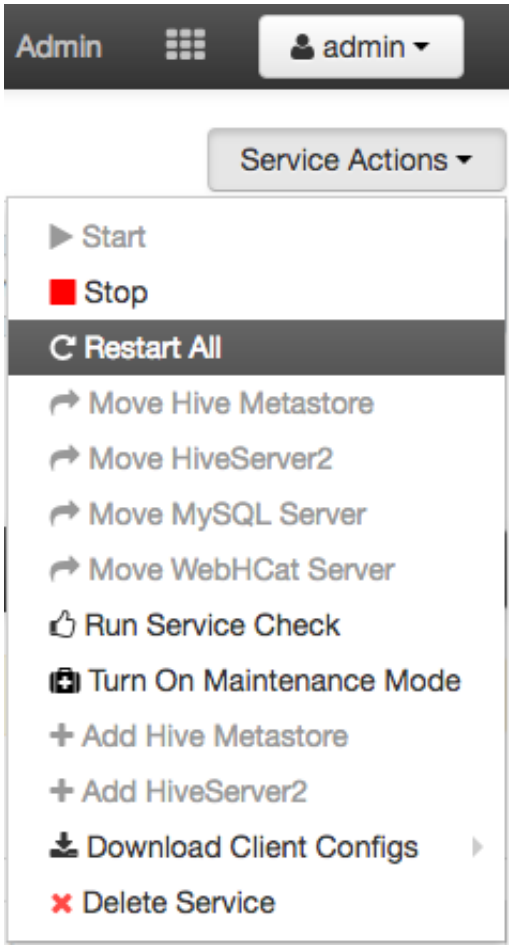
When enabling Hive LLAP, the **Run as end user instead of Hive user** slider on the **Settings** subtab has no effect on the Hive instance. If you set the slider to **`True`**, this property switches from Hive user to end user *only when you run Hive in batch-processing mode*.

11. Click the **Save** button near the top of the Ambari window.

12. If the **Dependent Configurations** window appears, review recommendations and adjust if you know settings need to be changed for your environment.

13. Click **Service Actions** > **Restart All**.

Figure 1.5. Restart All in Top Right Corner of Ambari Window



Next Steps

Connect Clients to a Dedicated HiveServer2 Endpoint [15]

Tip

Hive View 2.0 in Ambari integrates with the general availability release of Hive LLAP. If you plan to use Hive View 2.0 with a Hive LLAP instance, ensure that the Use Interactive Mode property of Manage Ambari Views is set to **`true`**. See Settings and Cluster Configuration of the *Ambari Views Guide*.

1.2.3. Enable Hive LLAP: Advanced Setup

About this Task

If you are a more advanced user of Hive LLAP and want to use a customized query queue rather than the default **llap** queue, then use the following procedure to enable interactive queries.

Prerequisites

- Installation of Ambari 2.5.x
- The Hive Service and other interdependencies as prompted in Ambari must be running.
- Your customized interactive query queue must be set up. For more information, see the Capacity Scheduler chapter of the Hortonworks *YARN Resource Management Guide*.
- Complete the tasks in the Queues for Hive LLAP Sites section.
- YARN preemption must be enabled and configured as documented in the Enabling YARN Preemption for Hive LLAP section above.

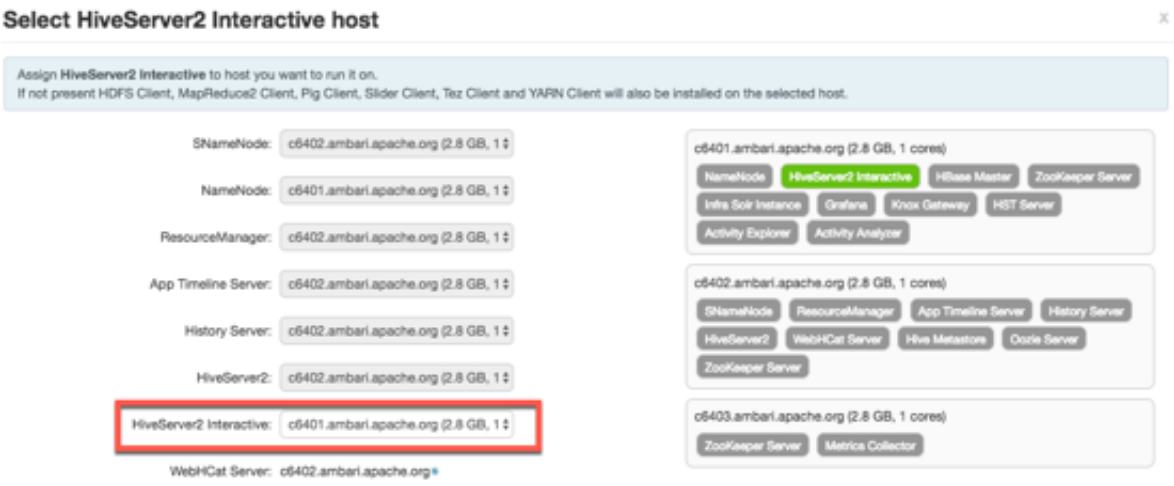
Steps

1. Select the **Hive** service in the Ambari dashboard.
2. Click the **Configs** tab.
3. In the **Settings** tab, locate the **Interactive Query** section and set the **Enable Interactive Query** slider to **Yes**.

Figure 1.6. Enable Interactive Query Toggle on the Settings Tab

4. Select the server to host the HiveServer2 Interactive instance in the **HiveServer2 Interactive** field. In most cases, you can accept the default server host assignment.

Figure 1.7. Select HiveServer2 Interactive Host Window



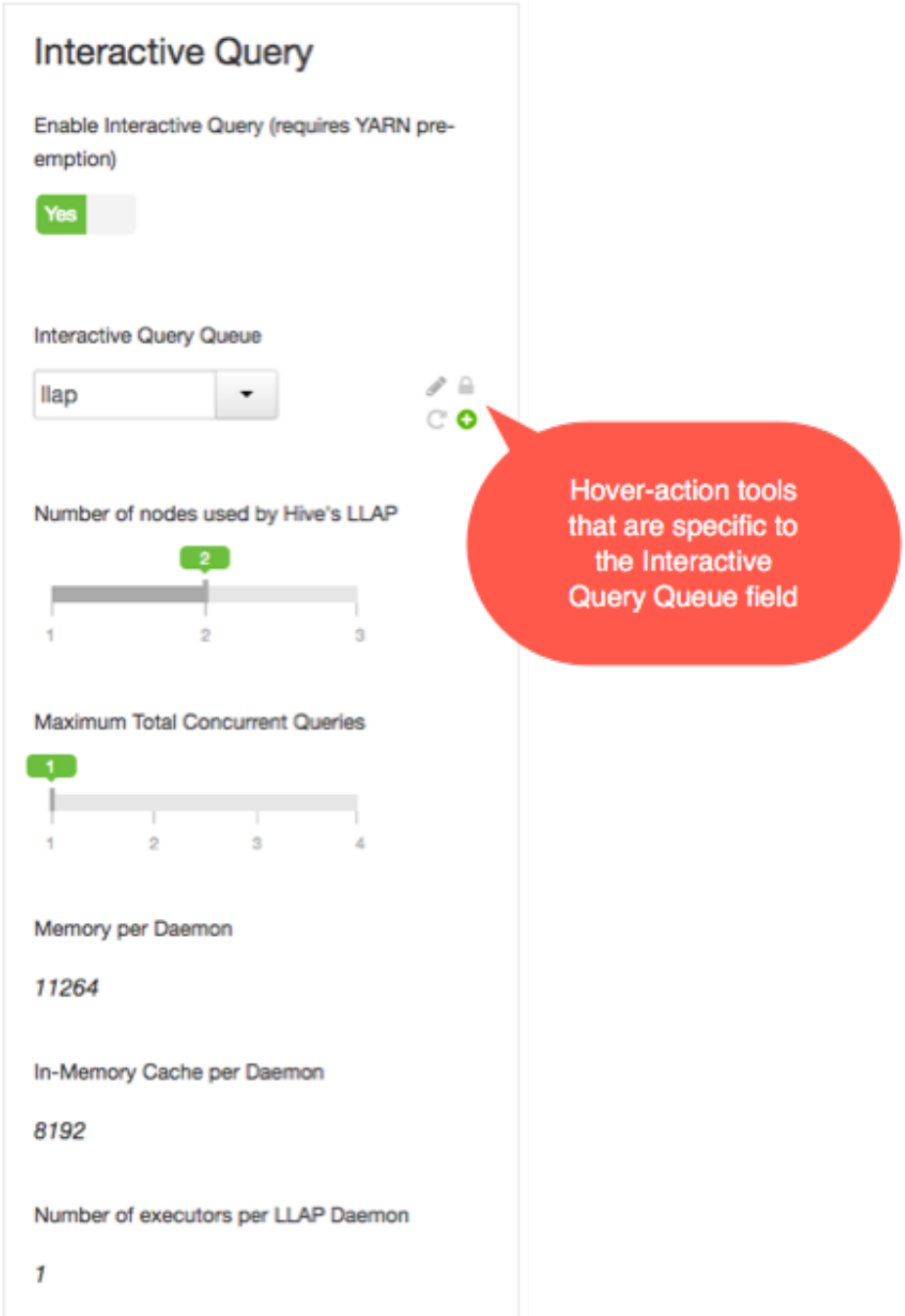
5. Select a predefined queue to use for the Hive LLAP cluster.

   a. Hover over the **Interactive Query Queue** field to display the hover-action tools, as illustrated in the following screenshot.

   b. Click the Edit (pencil icon) hover action to make the **Interactive Query Queue** field a drop-down list.

   c. Select the queue for Hive LLAP. This setting dedicates all the LLAP daemons and all the YARN ApplicationMasters of the system to the single, specified queue.

Figure 1.8. Enabled Interactive Query Configuration Settings



Important

Hover-action tools also appear when you move your pointer to hover over other editable elements of the Ambari window.

6. Set the **Number of nodes used by Hive LLAP** slider to the number of cluster nodes on which to run Hive LLAP. LLAP automatically deploys to the nodes, and you do not need to label the nodes.

7. Set the **Maximum Total Concurrent Queries** slider to the maximum number of concurrent Hive LLAP queries to run. The Ambari wizard calculates and displays a range of values in the slider based on the number of nodes that you selected and the number of CPUs in the Hive LLAP cluster. If you want to set the value outside the slider range, move your pointer over the field to enable the hover actions and select the Override tool.

8. Review the following settings, which are autogenerated for informational purposes only. (No interactive elements allow you to directly change the values.)

   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 executors per daemon: for example, the number of fragments that can execute in parallel on a daemon

9. Review the property settings outside the **Interactive Query** section of the window to learn how the Hive LLAP instance is configured. The Ambari wizard calculates appropriate values for most other properties on the **Settings** tab, based on the configurations in the **Interactive Query** section of the window.

   Important

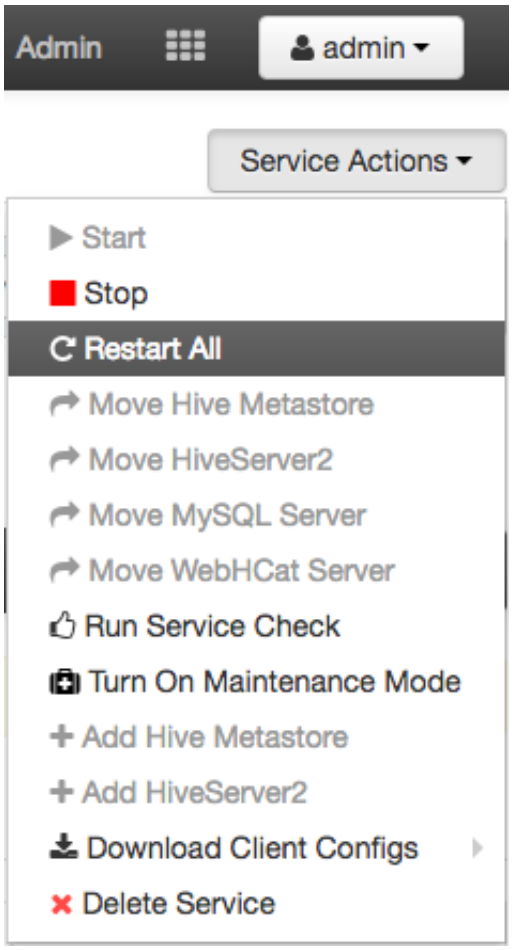
   When enabling Hive LLAP, the **Run as end user instead of Hive user** slider on the **Settings** tab has no effect on the Hive instance. If you set the slider to `True`, this property switches from Hive user to end user *only when you run Hive in batch-processing mode*.

10. Click the **Save** button near the top of the Ambari window.

11. If the **Dependent Configurations** window appears, review recommendations and adjust if you know settings need to be changed for your environment.

12. Click **Service Actions** > **Restart All**.

Figure 1.9. Restart All in Top Right Corner of Ambari Window



Next Steps

Connect Clients to a Dedicated HiveServer2 Endpoint [15]

Tip

Hive View 2.0 in Ambari integrates with the general availability release of Hive LLAP. If you plan to use Hive View 2.0 with a Hive LLAP instance, ensure that the Use Interactive Mode property of Manage Ambari Views is set to **`true`**. See Settings and Cluster Configuration of the *Ambari Views Guide*.

1.2.4. Connect Clients to a Dedicated HiveServer2 Endpoint

About this Task

Hortonworks supports Hive JDBC drivers that enable you to connect to HiveServer2 so that you can query, analyze, and visualize data stored in the Hortonworks Data Platform. In this task, you get the autogenerated HiveServer2 JDBC URL so that you can connect your client to the Hive LLAP instance.

Important

Do not use Hive CLI as your JDBC client for Hive LLAP queries.

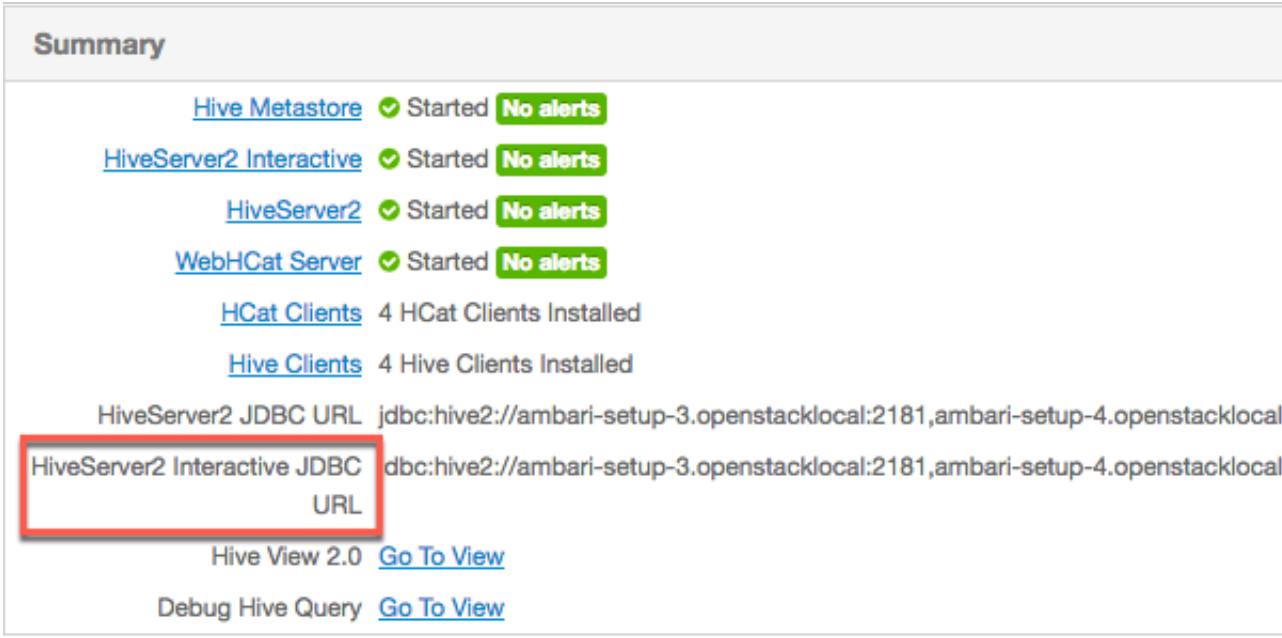
Prerequisite

Complete setup of Hive LLAP with Ambari, including restarting the Hive Service after saving the Enable Interactive Query settings.

Steps

1. Select the **Hive** service in the Ambari dashboard.
2. Click the **Summary** tab.
3. Use the clipboard icon to the right of the **HiveServer2 Interactive JDBC URL** value to copy the URL.

Figure 1.10. Summary Tab with the HiveServer2 JDBC URLs



4. Paste the URL into a JDBC client that you use to query the Hive EDW. For example, the client could be a BI tool or Beeline.

Next Steps

You can run your queries in the client. Hive LLAP should be booted and ready to use.

If query performance is too slow, see the following chapters of this guide.

1.3. Multiple Hive LLAP Instances in a Single HDP Cluster

If you want to deliver LLAP to multiple tenants that use a single HDP cluster, you can clone an existing HiveServer2 Interactive configuration of an Ambari-deployed LLAP instance and adjust the configuration to effectively create a new LLAP instance on the same cluster. With the following steps you can manipulate the HiveServer2 Interactive configurations so that you can run a separate LLAP instance on the same cluster. The steps can be repeated to create more than one additional LLAP instance. Because each LLAP instance runs on its own YARN queue, you can run queries through multiple LLAP instances on a single cluster simultaneously.

Procedure 1.1. Setting up Multiple LLAP Instances on the Same HDP Cluster

Prerequisites

- Ambari-deployed Hive LLAP instance.
- A separate YARN queue for a new LLAP instance with enough capacity to launch a Hive LLAP application, plus a sufficient number of Tez Application Masters to query the Hive LLAP instance.

1. In Ambari ensure that the **Enable Interactive Query** slider is set to **Yes**. This setting enables HiveServer2 Interactive.
2. Access the HiveServer2 Interactive node running the pre-existing LLAP instance:
   a. Select the **Hosts** tab of Ambari.
   b. Use the search field to locate the HiveServer2 Interactive node.
   c. On a command-line, log in to the node as root.
3. Clone the configuration of the HiveServer2 Interactive node. Choose a name for the clone that identifies the nature of the configuration and log files for the new Hive LLAP setup. The steps of this procedure assume that a second Hive LLAP instance is created, so the name myhive2 is used continuously in the following examples.
   - Run the following commands:

     HIVE_INSTANCE_NAME=myhive2

     mkdir -p /etc/hive2/conf.${HIVE_INSTANCE_NAME}

     cd /etc/hive2/conf.${HIVE_INSTANCE_NAME}

     cp -p /etc/hive2/2.6.1*/0/conf.server/*

     mkdir -p /var/log/hive.${HIVE_INSTANCE_NAME}

     chown hive:hadoop /var/log/hive.${HIVE_INSTANCE_NAME}
4. Adjust the configuration files for the new HiveServer2 Interactive instance:

a. Change HIVE_CONF_DIR in `/etc/hive2/conf.${HIVE_INSTANCE_NAME}/hive-env.sh` to the new configuration path.

Example: **HIVE_CONF_DIR=/etc/hive2/conf.myhive2**

b. Change the values of the following properties in the `/etc/hive2/conf.${HIVE_INSTANCE_NAME}/hive-site.xml` file:

- hive.server2.tez.default.queues: change the value to the queue where container jobs run or to default. If you do not change the value of hive.server2.tez.default.queues, there is not enough capacity for jobs to run in the environment that you are creating.
- hive.server2.zookeeper.namespace: change the value to hiveserver2-${HIVE_INSTANCE_NAME}, substituting the variable with your real value. For example, hiveserver2-myhive2.
- hive.server2.thrift.port: replace the default value (10000 in Hive 1; 10500 in Hive 2) with a number that does not clash with the port number of an existing HiveServer2 instance
- hive.server2.thrift.http.port: replace the default value (10001 in Hive 1; 10501 in Hive 2) with a number that does not clash with the port number of an existing HiveServer2 instance
- hive.server2.webui.port: replace the default value (10002 in Hive 1; 10502 in Hive 2) with a number that does not clash with the port number of an existing HiveServer2 instance

c. Change the value of property.hive.log.dir in the `hive-log4j2.properties` file to the new HiveServer2 configuration path.

Example: **property.hive.log.dir = /var/log/hive.myhive2**

5. Create a Hive LLAP package for your application by following the steps in Installing LLAP on an Unsecured Cluster or Installing LLAP on a Secured Cluster, depending on the environment that you have. Ensure that the *--queue* option is included in the command and that it directs the execution to the YARN queue to be used for the new LLAP instance. The following is an example of an executing command:

```
hive --service llap --name llap_extra --queue my_queue --instances 5 \
--cache 50000m --xmx 50000m --size 100000m --executors 12 --loglevel WARN
 \
--args "-XX:+UseG1GC -XX:+ResizeTLAB -XX:+UseNUMA  -XX:-ResizePLAB"
```

6. Launch LLAP using the `run.sh` file that was created after you completed Step 5.

Example:`sh llap-slider-${DATE}/run.sh`

7. Confirm the Hive LLAP instance is running. Wait a moment for the LLAP application to start within YARN. After startup of the application you can validate that the LLAP instance is running properly using the **llapstatus** command.

Example of llapstatus command:

```
/usr/hdp/current/hive-server2-hive2/bin/hive --service llapstatus -w -r 0.
8 -i 2 -t 200
```

If LLAP started successfully, you see output similar to the following message:

```
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/hdp/2.6.3.0-159/hive2/lib/log4j-
slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/hdp/2.6.3.0-159/hadoop/lib/slf4j-
log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an
 explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.
Log4jLoggerFactory]
WARN conf.HiveConf: HiveConf of name hive.llap.daemon.service.ssl does not
 exist
WARN conf.HiveConf: HiveConf hive.llap.daemon.vcpus.per.instance expects
 INT type value

LLAPSTATUS WatchMode with timeout=200 s
------------------------------------------------------------------------------
LLAP Application running with ApplicationId=application_1507473809953_0012
------------------------------------------------------------------------------
LLAP Application running with ApplicationId=application_1507473809953_0012
------------------------------------------------------------------------------

{
  "amInfo" : {
    "appName" : "llap1",
    "appType" : "org-apache-slider",
    "appId" : "application_1507473809953_0012",
    "containerId" : "container_e01_1507473809953_0012_01_000001",
    "hostname" : "ctr-e134-1499953498516-207726-02-000006.hwx.site",
    "amWebUrl" : "http://ctr-e134-1499953498516-207726-02-000006.hwx.
site:44549/"
  },
  "state" : "RUNNING_ALL",
  "originalConfigurationPath" : "hdfs://ctr-
e134-1499953498516-207726-02-000010.hwx.site:8020/user/hive/.slider/
cluster/llap1/snapshot",
  "generatedConfigurationPath" : "hdfs://ctr-
e134-1499953498516-207726-02-000010.hwx.site:8020/user/hive/.slider/
cluster/llap1/generated",
  "desiredInstances" : 1,
  "liveInstances" : 1,
  "launchingInstances" : 0,
  "appStartTime" : 1507542670971,
  "runningThresholdAchieved" : true,
  "runningInstances" : [ {
    "hostname" : "ctr-e134-1499953498516-207726-02-000005.hwx.site",
    "containerId" : "container_e01_1507473809953_0012_01_000002",
    "logUrl" : "http://ctr-e134-1499953498516-207726-02-000005.hwx.
site:54321/node/containerlogs/container_e01_1507473809953_0012_01_000002/
hive",
    "statusUrl" : "http://ctr-e134-1499953498516-207726-02-000005.hwx.
site:15002/status",
    "webUrl" : "http://ctr-e134-1499953498516-207726-02-000005.hwx.
site:15002",
    "rpcPort" : 44315,
```

```
    "mgmtPort" : 15004,
    "shufflePort" : 15551,
    "yarnContainerExitStatus" : 0
  } ]
}
```

8. Start the new HiveServer2 instance by running the following command:

```
export HIVE_CONF_DIR=/etc/hive2/conf.${HIVE_INSTANCE_NAME} \
/usr/hdp/current/hive-server2-hive2/bin/hive --service hiveserver2
```

For validation purposes, run this HiveServer2 instance in the foreground until you confirm it is functioning in the next step.

9. Validate the HiveServer2 instance started and check its version by running the **select version()** command.

Command example:

```
beeline \
    -u "jdbc:hive2://hs2host.example.com:port/" \
    -e "select version()"
```

Output example:

The following sample output assumes that the hive.server2.thrift.port property in Step 4 was set to 20500.

```
beeline -u "jdbc:hive2://ambari.example.com:20500/" -e "select version()"
Connecting to jdbc:hive2://ambari.example.com:20500/
Connected to: Apache Hive (version 2.1.0.2.6.1.0-129)
Driver: Hive JDBC (version 1.2.1000.2.6.1.0-129)
Transaction isolation: TRANSACTION_REPEATABLE_READ
DEBUG : Acquired the compile lock
INFO  : Compiling command(queryId=
hive_20170628220829_dbb26dfe-0f65-4492-8b8c-fa1469065f74): select
 version()
INFO  : We are setting the hadoop caller context from
 HIVE_SSN_ID:917c22bf-9998-413d-88c1-4b641391e060 to
 hive_20170628220829_dbb26dfe-0f65-4492-8b8c-fa1469065f74
INFO  : Semantic Analysis Completed
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:_c0,
 type:string, comment:null)], properties:null)
INFO  : Completed compiling command(queryId=
hive_20170628220829_dbb26dfe-0f65-4492-8b8c-fa1469065f74); Time taken: 0.
136 seconds
INFO  : We are resetting the hadoop caller context to
 HIVE_SSN_ID:917c22bf-9998-413d-88c1-4b641391e060
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Setting caller context to query id
 hive_20170628220829_dbb26dfe-0f65-4492-8b8c-fa1469065f74
INFO  : Executing command(queryId=
hive_20170628220829_dbb26dfe-0f65-4492-8b8c-fa1469065f74): select
 version()
INFO  : Resetting the caller context to
 HIVE_SSN_ID:917c22bf-9998-413d-88c1-4b641391e060
INFO  : Completed executing command(queryId=
hive_20170628220829_dbb26dfe-0f65-4492-8b8c-fa1469065f74); Time taken: 0.
033 seconds
```

```
INFO  : OK
DEBUG : Shutting down query select version()
+----------------------------------------------------------+--+
|                          _c0                             |
+----------------------------------------------------------+--+
| 2.1.0.2.6.1.0-129 rf65a7fce6219dbd86a9313bb37944b89fa3551b1  |
+----------------------------------------------------------+--+
1 row selected (0.345 seconds)
Beeline version 1.2.1000.2.6.1.0-129 by Apache Hive
Closing: 0: jdbc:hive2://ambari.example.com:20500/
```

10. You can also choose to use the ZooKeeper endpoint, based on the value of the hive.server2.zookeeper.namespace property.

    Example:

    ```
    beeline \
      -u "jdbc:hive2://ambari.example.com:2181/;serviceDiscoveryMode=
    zooKeeper;zooKeeperNamespace=hiveserver2-myhive2" \
      -e "select version()"
    ```

11. Run HiveServer2 in the background after validation.

    Command example:

    ```
    su - hive
    export HIVE_CONF_DIR=/etc/hive2/conf.${HIVE_INSTANCE_NAME}
    nohup /usr/hdp/current/hive2-server2/bin/hiveserver2 \
    -hiveconf hive.metastore.uris=" " -hiveconf hive.log.file=hiveserver2.log
     \
    >/var/log/hive/hiveserver2.out 2> /var/log/hive/hiveserver2err.log &
    ```

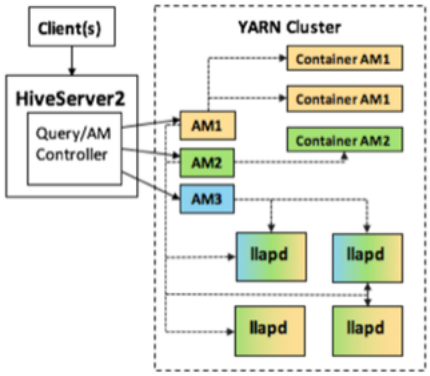
1.3.1. New HiveServer2 Instances in a Kerberized Cluster

The new HiveServer2 instance that was created in the procedure above points to the same keytab and uses the same service principal as the source instance that was cloned. If you need this instance to use a separate keytab or principal, make these additional changes in the `hive-site.xml` file.

2. Hive LLAP on Your Cluster

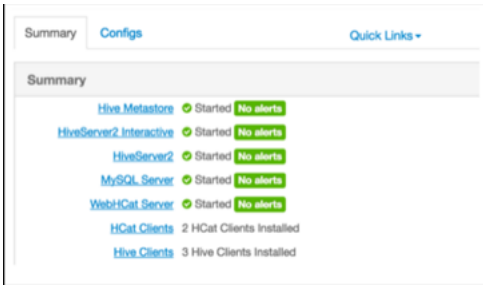
After setup, Hive LLAP is transparent to Apache Hive users and business intelligence tools. Interactive queries run on Apache Hadoop YARN as an Apache Slider application. You can monitor the real-time performance of the queries through the YARN ResourceManager Web UI or by using Slider and YARN command-line tools. Running through Slider enables you to easily open your cluster, share resources with other applications, remove your cluster, and flexibly utilize your resources. For example, you could 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.

Figure 2.1. LLAP on Your Cluster



On your cluster, an extra HiveServer2 instance is installed that is dedicated to interactive queries. You can see this HiveServer2 instance listed in the Hive Summary page of Ambari:

Figure 2.2. Hive Summary



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

Figure 2.3. ResourceManager Web UI

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:

Figure 2.4. Concurrency Setting

3. Best Practices Prior to Tuning Performance

Tip

Before tuning an Apache Hive system in depth, ensure that you adhere to the following best practices with your Hive deployment.

Use ORCFile format

Store all data in ORCFile format. See Maximizing Storage Resources for Hive.

Run the Hive EDW on Tez

Use Hive LLAP with Tez or the Hive on Tez execution engine rather than MapReduce.

Verify LLAP status for interactive queries

If you want to run interactive queries, ensure that the Hive LLAP engine is activated and configured. The best way to enable Hive LLAP or check if it is enabled is to use Ambari.

Check explain plans

Ensure queries are fully vectorized by examining their explain plans. See Optimizing the Hive Execution Engine for a conceptual explanation of different explain plans. Go directly to the Query Tab section of the Ambari Hive View 2.0 documentation to learn how to generate and interpret visual explain plans.

Run SmartSense

Use the SmartSense tool to detect common system misconfigurations. See the SmartSense documentation site for more information.

4. Connectivity and Admission Control

Creating and maintaining an environment for performant data analytics applications using a Hive EDW requires orchestrating several software components that reside on your cluster and using compatible client tools. The main pieces that concern the application developer and IT or DevOps staff are the following:

- HiveServer2: A service that connects your client application to the Hive EDW.
- YARN: A system resource for queuing data queries.
- Cost-Based Optimizer: An enhanced queuing mechanism of Hive.
- Apache Tez: An application framework for running high-performance batch and interactive data applications.
- For interactive and sub-second queries: Hive LLAP daemons. The daemons manage resources across all YARN nodes, rather than relying on separate Tez sessions on each node of the cluster.

HiveServer2, YARN, and Tez are components that work together to "intelligently" queue incoming queries on your Hive data set to minimize latency of returned results.

HiveServer2 is one of several architectural components for admission control. Admission control is designed to minimize query response time while enabling high concurrency. It operates by scaling the Hive processing of concurrent queries to the available system resources while removing the traditional launch time associated with MapReduce or Tez applications by maintaining long-living sessions. Admission control is akin to "connection pooling" in RDBMS databases.

To optimize Hive performance, configuration parameter settings that affect admission control must be optimized in line with your needs and system resources.

This chapter focuses on what you need to know about the components listed above to ensure clients connect to the Hive data warehouse and receive query results with high performance. To achieve optimal results, you also need to tune the data warehouse infrastructure so that it can handle concurrent queries in the way that comes closest to meeting your priorities.

4.1. HiveServer2

HiveServer2 is a server interface that enables remote clients to execute queries against Hive and retrieve the results using a JDBC or ODBC connection. For a client, you can use one of various BI tools (for example, Microstrategy, Tableau, and BusinessObjects) or another type of application that can access Hive over a JDBC or ODBC connection. In addition, you can also use a command-line tool, such as Beeline, that uses JDBC to connect to Hive.

Important

Do not use the Hive command-line interface (CLI). Instead, use the Beeline command-line shell or another JDBC CLI.

An embedded metastore, which is different from the MetastoreDB, also runs in HiveServer2. This metastore performs the following tasks:

- Get statistics and schema from the MetastoreDB
- Compile queries
- Generate query execution plans
- Submit query execution plans
- Return query results to the client

4.1.1. Sizing HiveServer2 Heap Memory

The following are general recommendations for sizing heap memory of a HiveServer2 instance:

- 1 to 20 concurrent executing queries: Set to 6 GB heap size.
- 21 to 40 concurrent executing queries: Set to 12 GB heap size.
- More than 40 concurrent executing queries: Create a new HiveServer2 instance. See Multiple HiveServer2 Instances for Different Workloads for how to add a HiveServer2 instance.

4.1.2. HiveServer2 Interactive UI

Important

The HiveServer2 Interactive UI functions only with clusters that have LLAP enabled.

The HiveServer2 Interactive UI monitors and displays heap, system, and cache metrics of each Hive LLAP node.

Tip

The HiveServer2 Interactive UI enables you to view executing queries in real time, a recent history of queries, and access running LLAP daemons. The Tez View provides diagnostics for debugging queries that executed or attempted to execute in the past.

From the **Quick Links** menu of Ambari, shown in the following figure, you can open the HiveServer2 Interactive UI.

Figure 4.1. Quick Links

4.1.3. Multiple HiveServer2 Instances for Different Workloads

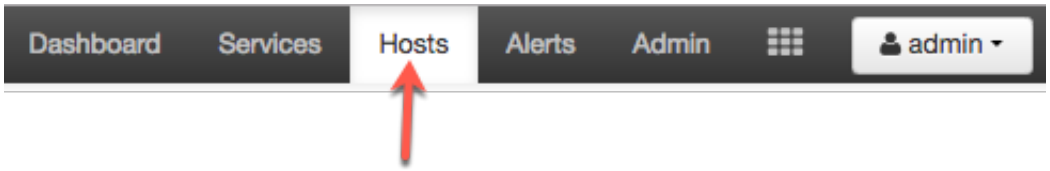
Multiple HiveServer2 instances can be used for:

- Load-balancing and high availability using ZooKeeper
- Running multiple applications with different settings

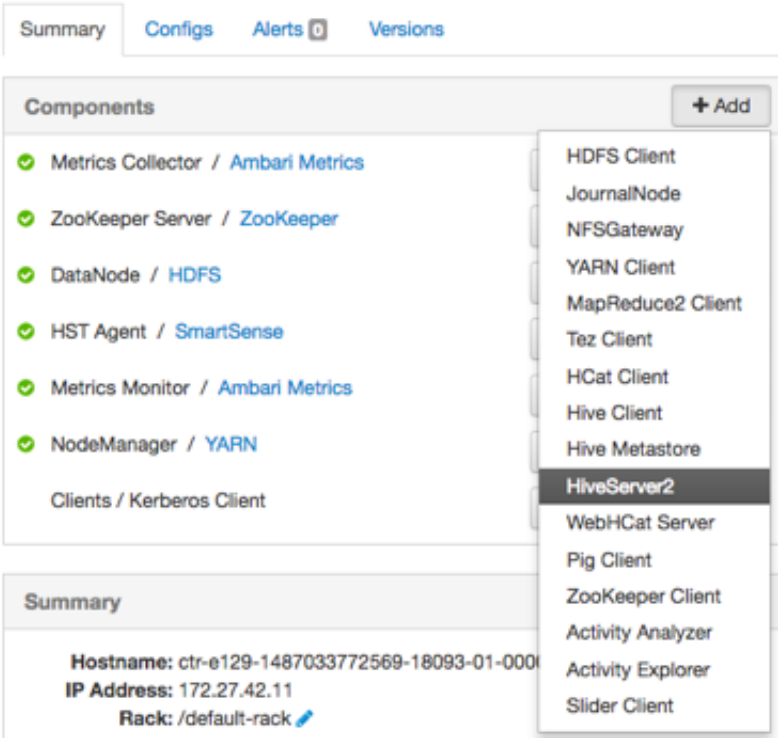
Because HiveServer2 uses its own settings file, using one for ETL operations and another for interactive queries is a common practice. All HiveServer2 instances can share the same MetastoreDB.

Adding a HiveServer2 Instance to Your Cluster

1. In Ambari, select the **Hosts** window



2. Click the name of the host node where you want to create the HiveServer2 instance.
3. On the **Summary** tab, click the **Add button** and select **HiveServer2**.



4.1.4. Security

HiveServer2 performs standard SQL security checks when a query is submitted, including connection authentication. After the connection authentication check, the server runs

authorization checks to make sure that the user who submits the query has permission to access the databases, tables, columns, views, and other resources required by the query. Hortonworks recommends that you use SQLStdAuth or Ranger to implement security. Storage-based access controls, which is suitable for ETL workloads only, is also available.

4.2. Workload Management with YARN Capacity Scheduler Queues

YARN allocates Hadoop cluster resources among users and groups with Capacity Scheduler queues. The Hive queries that are submitted from your data analytics applications compose just one set of the competing resource demands from different Hortonworks Data Platform (HDP) components.

You can configure the Capacity Scheduler queues to scale Hive batch and LLAP workloads as needed for your environment. However, the queue configuration in YARN for batch processing is different from the YARN configuration for Hive LLAP.

4.2.1. Queues for Batch Processing

Capacity Scheduler queues can be used to allocate cluster resources among users and groups. These settings can be accessed from **Ambari > YARN > Configs > Scheduler** or in **conf/capacity-scheduler.xml**.

The following configuration example demonstrates how to set up Capacity Scheduler queues. This example separates short- and long-running queries into two separate queues:

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

The following **capacity-scheduler.xml** settings are used to implement this configuration:

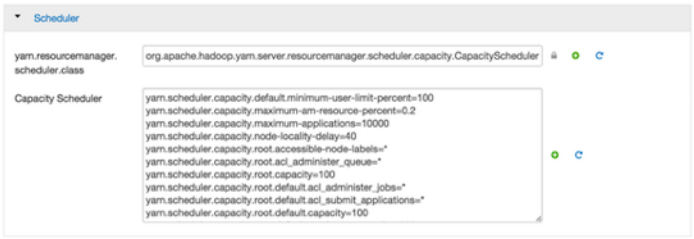
```
yarn.scheduler.capacity.root.queues=hive1,hive2
yarn.scheduler.capacity.root.hive1.capacity=50
yarn.scheduler.capacity.root.hive2.capacity=50
```

Configure usage limits for these queues and their users with the following settings:

```
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
```

Setting **maximum-capacity** to 50 restricts queue users to 50% of the queue capacity with a hard limit. If the **maximum-capacity** is set to more than 50%, the queue can use more than its capacity when there are other idle resources in the cluster. However, any user can use only the configured queue capacity. 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.

Figure 4.2. YARN Capacity Scheduler



This example is a basic introduction to queues. For more detailed information on allocating cluster resources using Capacity Scheduler queues, see the "Capacity Scheduler" section of the *YARN Resource Management Guide*.

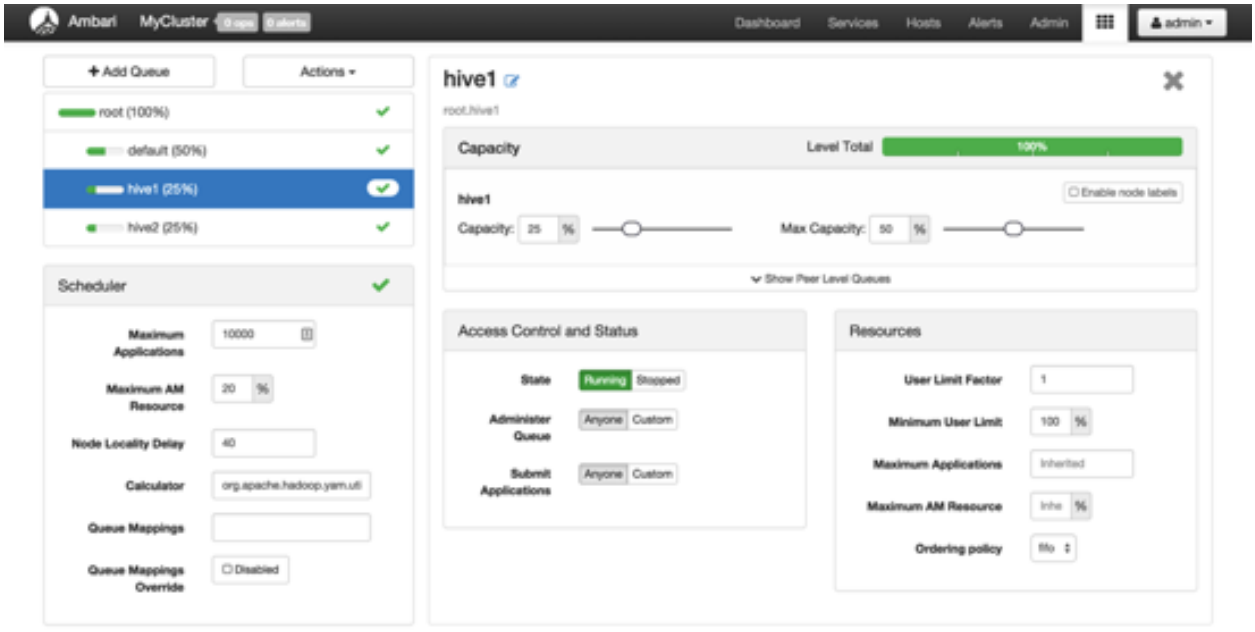
Setup Using the Ambari Capacity Scheduler View

If you are using Ambari 2.1 or later, queues can be set up using the Ambari Capacity Scheduler View as shown in the following image:

1. In Ambari, navigate to the administration page.
2. Click **Views > CAPACITY-SCHEDULER > <your_view_name>**, and then click **Go to instance** at the top of your view page.
3. In your view instance page, select the queue you want to use or create a queue. See the *Ambari Views Guide*.

   To create the scenario that is shown in the following screen capture, select the **root** queue and add **hive1** and **hive2** at that level.

Figure 4.3. Ambari Capacity Scheduler View



4.2.2. Queues in Hive LLAP Sites

If you accept the default **llap** queue of the Hive LLAP Service in Ambari, then no manual configuration of the YARN Capacity Scheduler is required. But if you prefer to create and customize the workload queue for interactive queries, then you need to complete the following task *before* enabling and configuring Hive LLAP in Ambari.

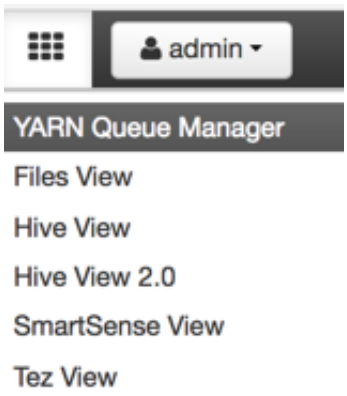
Important

If you are an advanced Hive user and want to launch Hive LLAP with a customized queue, complete the following task before enabling Hive LLAP. Do not complete the following task if plan to use the default **llap** queue that can be deployed automatically by starting the Hive LLAP Service in Ambari.

Setup of YARN for a Non-Default LLAP Queue

1. Create and configure a queue with the YARN Capacity Scheduler.
2. Open the Queue Manager View of Ambari:

Figure 4.4. YARN Queue Manager on the Views Menu



3. Select the queue that should be used by Hive LLAP.
4. In the **Resources** pane, set the **Priority** field with a number that is larger than the priority value of the other queues. The default value of the Priority field after a queue is created is zero.

5. Using the Cost-Based Optimizer to Enhance Performance

Hive's Cost-Based Optimizer (CBO) is a core component in Hive's query processing engine. Powered by Apache Calcite, the CBO optimizes and calculates the cost of various plans for a query.

The main goal of a CBO is to generate efficient execution plans by examining the tables and conditions specified in the query, ultimately cutting down on query execution time and reducing resource utilization. After parsing, a query gets converted to a logical tree (Abstract Syntax Tree) that represents the operations that the query must perform, such as reading a particular table or performing an inner JOIN.

Calcite applies various optimizations such as query rewrite, JOIN reordering, and deriving implied predicates and JOIN elimination to produce logically equivalent plans. The current model prefers bushy plans for maximum parallelism. Each logical plan is assigned a cost based in number of distinct value based heuristics.

Calcite has an efficient plan pruner that can select the cheapest query plan. The chosen logical plan is then converted by Hive to a physical operator tree, optimized and converted to Tez jobs, and then executed on the Hadoop cluster.

5.1. Enabling Cost-Based Optimization

About this Task

Turning on CBO is highly recommended.

Prerequisite

You must have administrator previleges.

Steps

1. In Ambari, open **Services** > **Hive** > **Configs** tab.
2. Refer to the following table for the properties that enable CBO and assist with generating table statistics, along with the required property settings.

   You can view the properties by either of these methods:

   Type each property name in the Filter field in the top right corner.
   Open the **General**, **Advanced hive-env**, etc., sections and scan the lists of each category.
3. Click **Save**.
4. If prompted to restart, restart the Hive Service.

Table 5.1. CBO Configuration Parameters

Configuration Parameter	Setting to Enable CBO	Description
`hive.cbo.enable`	**`true`**	Enables cost-based query optimization.

Configuration Parameter	Setting to Enable CBO	Description
`hive.stats.autogather`	**`true`**	Enables automated gathering of table-level statistics for newly created tables and table partitions, such as tables created with the INSERT OVERWRITE statement. The parameter does not produce column-level statistics, such as those generated by CBO. If disabled, administrators must manually generate the table-level statistics for newly generated tables and table partitions with the ANALYZE TABLE statement.

5.2. Statistics

Tip

Gather both column and table statistics for best query performance.

Column and table statistics must be calculated for optimal Hive performance because they are critical for estimating predicate selectivity and cost of the plan. In the absence of table statistics, Hive CBO does not function. Certain advanced rewrites require column statistics.

Ensure that the configuration properties in the following table are set to `true` to improve the performance of queries that generate statistics. You can set the properties using Ambari or by customizing the `hive-site.xml` file.

Configuration Parameter	Setting to Enable Statistics	Description
`hive.stats.fetch.column.stats`	**`true`**	Instructs Hive to collect column-level statistics.
`hive.compute.query.using.stats`	**`true`**	Instructs Hive to use statistics when generating query plans.

5.2.1. Generating Hive Statistics

The ANALYZE TABLE command generates statistics for tables and columns. The following lines show how to generate different types of statistics on Hive objects.

Gathering table statistics for non-partitioned tables

```
ANALYZE TABLE [table_name] COMPUTE STATISTICS;
```

Gathering table statistics for partitioned tables

```
ANALYZE TABLE [table_name] PARTITION(partition_column) COMPUTE STATISTICS;
```

Gathering column statistics for the entire table

```
ANALYZE TABLE [table_name] COMPUTE STATISTICS for COLUMNS
 [comma_separated_column_list];
```

Gathering statistics for the **`partition2`** column on a table partitioned on **`col1`** with key **`x`**

```
ANALYZE TABLE partition2 (col1="x") COMPUTE STATISTICS for COLUMNS;
```

5.2.2. Viewing Generated Statistics

Use the DESCRIBE statement to view statistics generated by CBO. Include the EXTENDED keyword if you want to include statistics gathered when the `hive.stats.fetch.column.stats` and `hive.compute.query.using.stats` properties are enabled.

- Viewing Table Statistics
  - Use the following syntax to view table statistics:

    ```
    DESCRIBE [EXTENDED] table_name;
    ```

    Note

    The EXTENDED keyword can be used only if the `hive.stats.autogather` property is enabled in the `hive-site.xml` configuration file.

  - The following example displays all statistics for the employees table:

    ```
    DESCRIBE EXTENDED employees;
    ```

  - If the table statistics are up-to-date, the output includes the following table parameter information:

    ```
    {\"BASIC_STATS\":\"true\",\" ...
    ```

- Viewing Column Statistics
  - Use the following syntax to view column statistics:

    ```
    DESCRIBE FORMATTED [db_name.]table_name.column_name;
    ```

  - The following example displays statistics for the region column in the employees table:

    ```
    DESCRIBE FORMATTED employees.region;
    ```

  - If the table statistics are up-to-date, the output includes the following table parameter information:

    ```
    COLUMN_STATS_ACCURATE
    ```

Note

See Statistics in Hive on the Apache website for more information.

5.3. SQL Optimization and Planning Properties

Ambari has a configuration wizard that automatically tunes some of the optimization- and planner-related configuration properties of Hive, Tez, and YARN.

Tip

In most cases, do not change the settings for properties that have *Auto-tuned* default settings listed in the following table. The values that are set for these properties are calculated by your cluster profile and rarely need to be overwritten.

Table 5.2. Settings for Optimization and Planning Properties

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

6. Optimizing the Hive Execution Engine

To maximize the data analytics capabilities of applications that query Hive, you might need to tune the Apache Tez execution engine. Tez is an advancement over earlier application frameworks for Hadoop data processing, such as MapReduce2 and MapReduce1. The Tez framework is required for high-performance batch workloads and for all interactive applications.

6.1. Explain Plans

When you use Hive for interactive queries, you can generate explain plans. An explain plan shows you the execution plan of a query by revealing the series of operations that occur when a particular query is run. By understanding the plan, you can determine if you want to adjust your application development.

For example, an explain plan might help you see why the query optimizer runs a query with a shuffle operation instead of a hash JOIN. With this knowledge, you might want to rewrite queries in the application so that they better align with user goals and the environment.

Hive in HDP can generate two types of explain plans. A *textual* plan, such as information printed in a CLI query editor, displays the execution plan in descriptive lines. A *graphical* plan, such as the Visual Explain feature of Hive Views in Ambari, shows the execution plan as a flow diagram. Learn more about Visual Explain Plans in the Query Tab documentation for Hive View 2.0.

6.2. Tuning the Execution Engine Manually

If you encounter subpar performance of your Hive queries after debugging them with Tez View and Hive View, then you might need to adjust Tez Service configuration properties.

6.2.1. Tune Tez Service Configuration Properties

About this Task

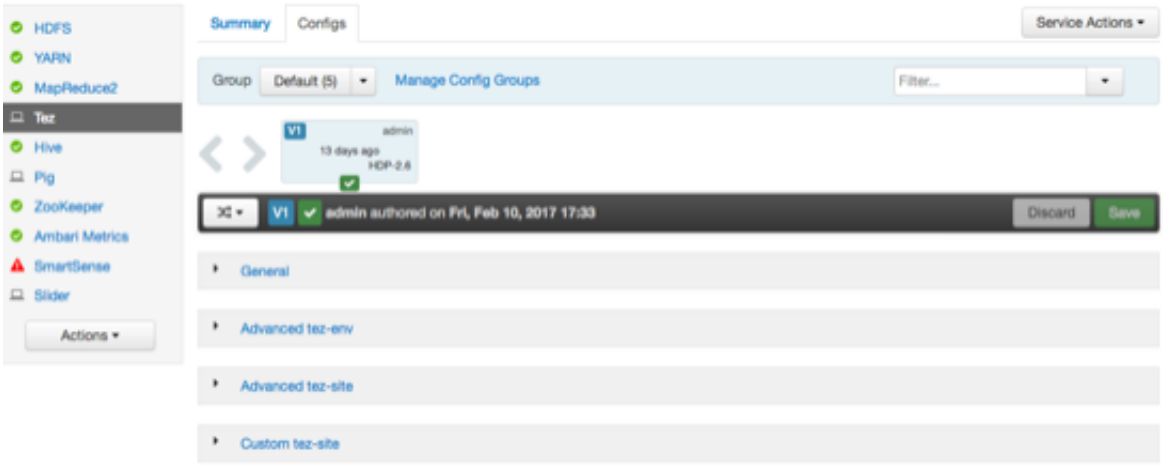
Important

Check and adjust the following property settings only if you think these execution engine properties degrade the performance of **Hive LLAP** queries.

Advanced users: If you want to add or configure a property that is not listed in the table below, open the **Custom tez-site** section of the **Configs** tab to enter or edit the custom property.

Steps

1. In Ambari, open **`Services`** > **`Tez`** > **Configs** tab.



2. Use the following table as a reference checklist.

Tip

Ambari automatically customizes the value for the *tez.am.resource.memory.mb* property to suit your cluster profile. Generally, you should not change the default value of this property at this stage if you are not changing resources on the cluster.

3. You can view the properties by either of these methods:

   Type each property name in the Filter field in the top right corner.
   Open the **General**, **Advanced tez-env**, etc., sections and scan the lists of each category.

4. Click **Save**.

5. If prompted to restart, restart the Tez Service.

Table 6.1. Settings for Execution Engine Properties

Property	Setting Guideline *If* Manual Configuration Is Needed	Default Value in Ambari
tez.am.resource.memory.mb	4 GB maximum for most sites	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	**2.0**

Property	Setting Guideline *If* Manual Configuration Is Needed	Default Value in Ambari
	Decrease for fewer reducers	
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**

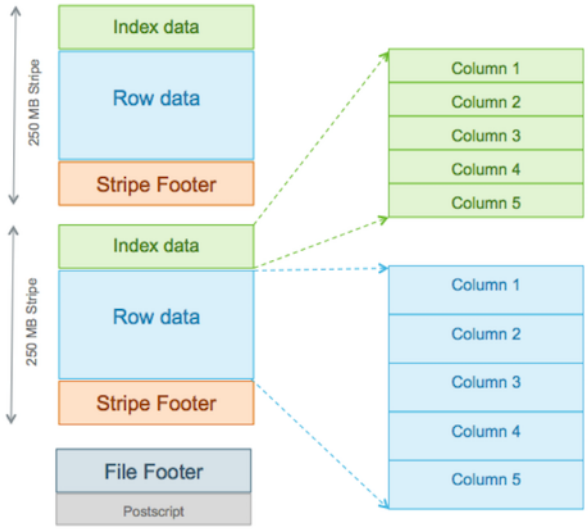
7. Maximizing Storage Resources

7.1. ORC File Format

The Optimized Row Columnar (ORC) file format provides the following advantages over many other file formats:

- **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.

Figure 7.1. ORC File Structure



Specifying the Storage Format as ORC

In addition, to specifying the storage format, you can also specify a compression algorithm for the table:

```
CREATE TABLE addresses (
   name string,
   street string,
   city string,
   state string,
   zip int
   ) STORED AS orc tblproperties ("orc.compress"="Zlib");
```

Note

Setting the compression algorithm is usually not required because your Hive settings include a default algorithm.

Switching the Storage Format to ORC

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

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

Ingestion as ORC

A common practice is to land 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.

Advanced Settings

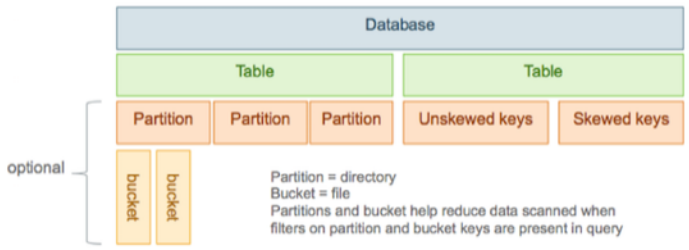
ORC has properties that usually do not need to be modified. However, for special cases you can modify the properties listed in the following table when advised to by Hortonworks Support.

Table 7.1. ORC Properties

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.

7.2. Designing Data Storage with Partitions and Buckets

Figure 7.2. Hive Data Abstractions



7.2.1. Partitioned Tables

In Hive, tables are often partitioned. *Partitions* map to physical directories on the filesystem. Frequently, tables are partitioned by date-time as incoming data is loaded into Hive each day. Large deployments can have tens of thousands of partitions.

Using partitions can significantly improve performance if a query has a filter on the partitioned columns, which can prune partition scanning to one or a few partitions that match the filter criteria. *Partition pruning* occurs directly when a partition key is present in the WHERE clause. Pruning occurs indirectly when the partition key is discovered during query processing. For example, after joining with a dimension table, the partition key might come from the dimension table.

Partitioned columns are not written into the main table because they are the same for the entire partition, so they are "virtual columns." However, to SQL queries, there is no difference:

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

To insert data into this table, the partition key can be specified for fast loading:

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

Without the partition key, the data can be loaded using dynamic partitions, but that makes it slower:

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, invoke a specific property that is designed specifically for this purpose. Enabling the property 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;
```

The virtual columns that are used as partitioning keys must be last. Otherwise, you must re-order the columns using a SELECT statement similar to the following:

```
INSERT INTO sale (xdate, state='CA')
SELECT id, amount, other_columns..., xdate
FROM staging_table
WHERE state='CA';
```

Tip

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.

7.2.2. Bucketed Tables

Tables or partitions can be further divided into *buckets*, which are stored as files in the directory for the table or the directories of partitions if the table is partitioned. Bucketing can optimize Hive's scanning of a data set that is the target of repeated queries.

When buckets are used with Hive tables and partitions, a common challenge is to maintain query performance while workload or data scales up or down. For example, you could have an environment where picking 16 buckets to support 1000 users operates smoothly, but a spike in the number of users to 100,000 for a day or two could create problems if the buckets and partitions are not promptly tuned. 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.

hive-site.xml setting for enabling table 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
```

Example of using HiveQL with bucketed data:

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:

- 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.

Tip

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

7.3. Supported Filesystems

While a Hive EDW can run on one of a variety of storage layers, HDFS and Amazon S3 are the most prevalently used and known filesystems for data analytics that run in the Hadoop stack. By far, the most common filesystem used for a public cloud infrastructure is Amazon S3.

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

Depending on your environment, you can tune the filesystem to optimize Hive performance by configuring compression format, stripe size, partitions, and buckets. Also, you can create Bloom filters for columns frequently used in point lookups.

8. Debugging Performance Issues

8.1. Debugging Hive Queries with Tez View

The Tez View of Ambari displays dashboards and visualizations that represent performance issues of problematic Hive queries. You can use the Tez View to troubleshoot what aspects of a Hive query are not running optimally or do not function at all. See Using Tez View in the Hortonworks *Apache Ambari Views Guide*for more information.

8.2. Viewing Metrics in Grafana

The Ambari Metrics System includes Grafana, with prebuilt dashboards for advanced visualization of cluster metrics. You can monitor the performance of the system through the Hive LLAP dashboards. The following dashboards are available. To learn more about these monitoring tools, see Hive LLAP Dashboards in the Hortonworks *Apache Ambari Operations Guide*.

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 accounted for only 8 nodes running.

If you find an issue in this dashboard, you can open the LLAP Daemon dashboard to see which node is having the problem.

Hive LLAP Daemon: Metrics that show operating status for a specific Hive LLAP Daemon