Cloudera Runtime 7.3.3

Configuring Apache Impala

Date published: 2020-11-30
Date modified: 2026-11-13



https://docs.cloudera.com/

Legal Notice

© Cloudera Inc. 2026. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

Configuring client access to Impala....4
Impala Startup Options for Client Connections....4
Impala Shell Tool....6
Impala Shell Configuration Options....6
Impala Shell Configuration File....9
Connecting to Impala Daemon in Impala Shell....10
Running Commands and SQL Statements in Impala Shell....12
Impala Shell Command Reference....13
Configuring ODBC for Impala....14
Configuring JDBC for Impala....14
Configuring Impyla for Impala....15
Connecting to Cloudera Data Hub....17
Connecting to Cloudera Data Hub Data Mart....18
Configuring Delegation for Clients....18
Spooling Impala Query Results....19
Configuring Impala....20
Configuring Impala for High Availability....22
Enabling Catalog and StateStore High Availability (HA)....22
Disabling Catalog and StateStore High Availability....24
Failure detection for Catalog and StateStore....25
Configuring Load Balancer for Impala....25
Running queries on system tables....31
Configuring the only coordinators request pool....31
Migrating Impala Catalog to another host....32

Configuring client access to Impala

Application developers have a number of options to interface with Impala.

The core development language with Impala is SQL. You can also use Java or other languages to interact with Impala through the standard JDBC and ODBC interfaces used by many business intelligence tools. For specialized kinds of analysis, you can supplement the Impala built-in functions by writing user-defined functions in C++ or Java.

You can connect and submit requests to the Impala through:

- The impala-shell interactive command interpreter
- The Data Explorer web-based user interface
- JDBC
- ODBC
- Impyla

Impala clients can connect to any Coordinator Impala Daemon (impalad) through HiveServer2 over HTTP or over the TCP binary or through Beeswax. All interfaces support Kerberos and LDAP for authentication to Impala. See below for the default ports and the Impala configuration field names to change the ports in Cloudera Manager.

Protocol	Default Port	Cloudera Manager Field to Specify an Alternate Port
HiveServer2 HTTP	28000	Impala Daemon HiveServer2 HTTP Port
HiveServer2 binary TCP	21050	Impala Daemon HiveServer2 Port
Beeswax	21000	Impala Daemon Beeswax Port

Impala Startup Options for Client Connections

Use the following flags to control client connections to Impala when starting Impala Daemon coordinator.

If a configuration field exists in Cloudera Manager, the field name is shown in parenthesis next to the flag name. If the Cloudera Manager interface does not yet have a form field for an option, the Advanced category page for each daemon includes one or more Safety Valve fields where you can enter option names directly.

--accepted_client_cnxn_timeout

Controls how Impala treats new connection requests if it has run out of the number of threads configured by --fe_service_threads.

If --accepted_client_cnxn_timeout > 0, new connection requests are rejected if Impala can't get a server thread within the specified (in seconds) timeout.

If --accepted_client_cnxn_timeout=0, i.e. no timeout, clients wait indefinitely to open the new session until more threads are available.

The default timeout is 5 minutes.

The timeout applies only to client facing thrift servers, i.e., HS2 and Beeswax servers.

--disconnected_session_timeout

When a HiveServer2 session has had no open connections for longer than this value, the session will be closed, and any associated queries will be unregistered.

Specify the value in hours.

The default value is 1 hour.

This flag does not apply to Beeswax clients. When a Beeswax client connection is closed, Impala closes the session associated with that connection.

--fe_service_threads (Impala Daemon Max Client)

Specifies the maximum number of concurrent client connections allowed. The default value is 128 with which 128 queries can run simultaneously.

If you have more clients trying to connect to Impala than the value of this setting, the later arriving clients have to wait for the duration specified by --accepted_client_cnxn_timeout. You can increase this value to allow more client connections. However, a large value means more threads to be maintained even if most of the connections are idle, and it could negatively impact query latency. Client applications should use the connection pool to avoid need for large number of sessions.

--idle_client_poll_time_s

The value of this setting specifies how frequently Impala polls to check if a client connection is idle and closes it if the connection is idle. A client connection is idle if all sessions associated with the client connection are idle.

By default, --idle_client_poll_time_s is set to 30 seconds.

If --idle_client_poll_time_s is set to 0, idle client connections stay open until explicitly closed by the clients.

The connection will only be closed if all the associated sessions are idle or closed. Sessions cannot be idle unless either the flag --idle_session_timeout or the IDLE_SESSION_TIMEOUT query option is set to greater than 0. If idle session timeout is not configured, a session cannot become idle by definition, and therefore its connection stays open until the client explicitly closes it.

--max_cookie_lifetime_s

Impala uses cookies for authentication when clients connect via HiveServer2 over HTTP. Use the --max_cookie_lifetime_s startup flag to control how long generated cookies are valid for.

Specify the value in seconds.

The default value is 1 day.

Setting the flag to 0 disables cookie support.

When an unexpired cookie is successfully verified, the user name contained in the cookie is set on the connection.

Each impalad uses its own key to generate the signature, so clients that reconnect to a different impa lad have to re-authenticate.

On a single impalad, cookies are valid across sessions and connections. See --cookie_secret_file to configure a consistent secret across impalad instances.

--beeswax_port (Impala Daemon Beeswax Port)

Specifies the port for clients to connect to Impala daemon via the Beeswax protocol.

You can disable the Beeswax end point for clients by setting the flag to 0.

--hs2_http_port (Impala Daemon HiveServer2 HTTP Port)

Specifies the port for clients to connect to Impala daemon over HTTP.

You can disable the HTTP end point for clients by setting the flag to 0.

To enable TLS/SSL for HiveServer2 HTTP endpoint, use --ssl_server_certificate and --ssl_privat e_key.

--hs2_port (Impala Daemon HiveServer2 Port)

Specifies the port for clients to connect to Impala daemon via the HiveServer2 protocol.

You can disable the binary HiveServer2 end point for clients by setting the flag to 0.

Impala Shell Tool

You can use the Impala shell tool (impala-shell) to set up databases and tables, insert data, and issue queries.

For ad-hoc queries and exploration, you can submit SQL statements in an interactive session. To automate your work, you can specify command-line options to process a single statement or a script file. The `impala-shell` accepts all the same SQL statements, plus some shell-only commands that you can use for tuning performance and diagnosing problems.

Cloudera Manager installs `impala-shell` automatically. You might install `impala-shell` manually on other systems not managed by Cloudera Manager, so that you can issue queries from client systems that are not also running the Impala daemon or other Apache Hadoop components.

Impala Shell Configuration Options

You can specify the following options when starting impala-shell to control how shell commands are executed. You can specify options on the command line or in the impala-shell configuration file.

Command-Line Option	Configuration File Setting	Explanation
-B or --delimited	write_delimited=true	Causes all query results to be printed in plain format as a delimited text file. Useful for producing data files to be used with other Hadoop components. Also useful for avoiding the performance overhead of pretty-printing all output, especially when running benchmark tests using queries returning large result sets. Specify the delimiter character with the --output_delimiter option. Store all query results in a file rather than printing to the screen with the -B option.
-E or --vertical	vertical=true	Causes all query results to print in vertical format. In this mode, impala-shell will print each row in the following style. • The first line will contain the line number followed by the row's columns in separate lines. Each column line will have a prefix with its name and a colon. To enable this mode, use the shell option '-E' or '--vertical', or 'set VERTICAL= true' in the interactive mode. To disable it in interactive mode, 'set VERTICAL=false'. NOTE: This vertical option will be disabled if the '-B' option or 'set WRITE_DELIMITED=true' is specified.
--connect_max_tries	--connect_max_tries=4	Sets the maximum number of attempts to connect to a coordinator. Currently, the maximum number of attempts to connect to a coordinator is hard coded to 4 in hs2-HTTP mode. From this release, you can configure the maximum number of attempts impala-shell can make using this option --connect_max_tries. The default value of this option is 4. After the number of attempts specified through this option, impala-shell returns with error "Not connected to Impala"
--hs2_fp_format	hs2_fp_format=<a Python-based format specification expression which will get parsed and applied to floating-pointcolumn values>	Sets the printing format specification for floating point values when using HS2 protocol. The default behaviour makes the values handled by Python's str() built-in method. Use '16G' to match Beeswax protocol's floating-point output format.
--http_cookie_names	http_cookie_names=KNOX_BACKEND-IMPALA	This parameter has to be set to include the cookie that Knox uses for stateful connections. This is what the Knox enableStickySession=true uses. When using impala-shell and http mode, you must update the client connection string to include the cookie being used for session persistence using the `-http_cookie_names`. This config is needed for Active-Active HA in Impala.
--live_progress	live_progress=true	Prints a progress bar showing roughly the percentage complete for each query. Information is updated interactively as the query progresses.
--disable_live_progress	live_progress=false	Disables live_progress in the interactive mode.

Command-Line Option	Configuration File Setting	Explanation
-b or --kerberos_host_fqdn	kerberos_host_fqdn= *LOAD-BALANCER-HOSTNAME*	If set, the setting overrides the expected hostname of the Impala daemon's Kerberos service principal. `impala-shell` will check that the server's principal matches this hostname. This may be used when impalad is configured to be accessed via a load-balancer, but it is desired for impala-shell to talk to a specific impalad directly.
--print_header	print_header=true	
-o *FILENAME* or --output_file *FILENAME*	output_file=*FILENAME*	Stores all query results in the specified file. Typically used to store the results of a single query issued from the command line with the -q option. Also works for interactive sessions; you see the messages such as number of rows fetched, but not the actual result set. To suppress these incidental messages when combining the -q and -o options, redirect stderr to /dev/null.
--output_delimiter= *CHARACTER*	output_delimiter=*CHARACTER*	Specifies the character to use as a delimiter between fields when query results are printed in plain format by the -B option. Defaults to tab ('\t'). If an output value contains the delimiter character, that field is quoted, escaped by doubling quotation marks, or both.
-p or --show_profiles	show_profiles=true	Displays the query execution plan (same output as the EXPLAIN statement) and a more detailed low-level breakdown of execution steps, for every query executed by the shell.
--profile_format= text | json | prettyjson	N/A	json and prettyjson output the JSON representation of each profile in a dense single-line form and in a human-readable multi-line form respectively.
-h or --help	N/A	Displays help information.
N/A	history_max=1000	Sets the maximum number of queries to store in the history file.
-i *HOSTNAME* or --impalad=*HOSTNAME*[:*PORTNUM*]	impalad=*HOSTNAME*[:*PORTNUM*]	Connects to the impalad daemon on the specified host. The default port of 21050 is assumed unless you provide another value. You can connect to any host in your cluster that is running impalad. If you connect to an instance of impalad that was started with an alternate port specified by the --fe_port flag, provide that alternative port.
-q *QUERY* or --query=*QUERY*	query=*QUERY*	Passes a query or other `impala-shell` command from the command line. The `impala-shell` interpreter immediately exits after processing the statement. It is limited to a single statement, which could be a SELECT, CREATE TABLE, SHOW TABLES, or any other statement recognized in impala-shell. Because you cannot pass a USE statement and another query, fully qualify the names for any tables outside the default database. (Or use the -f option to pass a file with a USE statement followed by other queries.)
-f *QUERY_FILE* or --query_file=*QUERY_FILE*	query_file=*PATH_TO_QUERY_FILE*	Passes a SQL query from a file. Multiple statements must be semicolon (;) delimited.
-k or --kerberos	use_kerberos=true	Kerberos authentication is used when the shell connects to impalad. If Kerberos is not enabled on the instance of impalad to which you are connecting, errors are displayed.
--query_option= "*OPTION=VALUE* -Q "*OPTION=VALUE*"	Header line [impala.query_options], followed on subsequent lines by *OPTION=VALUE*, one option per line.	Sets default query options for an invocation of the `impala-shell` command. To set multiple query options at once, use more than one instance of this command-line option. The query option names are not case-sensitive.

Command-Line Option	Configuration File Setting	Explanation
-s *KERBEROS_SERVICE_NAME* or --kerberos_service_name= *NAME*	kerberos_service_name=*NAME*	Instructs impala-shell to authenticate to a particular impalad service principal. If a *KERBEROS_SERVICE_NAME* is not specified, impala is used by default. If this option is used in conjunction with a connection in which Kerberos is not supported, errors are returned.
-V or --verbose	verbose=true	Enables verbose output.
--quiet	verbose=false	Disables verbose output.
-v or --version	version=true	Displays version information.
-c	ignore_query_failure=true	Continues on query failure.
-d *DEFAULT_DB* or --database=*DEFAULT_DB*	default_db=*DEFAULT_DB*	Specifies the database to be used on startup. Same as running the USE statement after connecting. If not specified, a database named DEFA ULT is used.
--ssl	ssl=true	Enables TLS/SSL for `impala-shell`.
--http_socket_timeout_s	http_socket_time out_s=*HTTP_SOCKET_TIMEOUT_S*	Sets the timeout in seconds after which the socket will time out if the associated operation cannot be completed. Set to None to disable any timeout. This configurable option is only supported for hs2-http mode and the DEFAULT is NONE.
PATH_TO_CERTIFICATE	ca_cert=*PATH_TO_CERTIFICATE*	The local pathname pointing to the third-party CA certificate, or to a copy of the server certificate for self-signed server certificates. If --ca_cert is not set, `impala-shell` enables TLS/SSL, but does not validate the server certificate. This is useful for connecting to a known-good Impala that is only running over TLS/SSL, when a copy of the certificate is not available (such as when debugging customer installations).
-l	use_ldap=true	Enables LDAP authentication.
-u	user=*USER_NAME*	Supplies the username, when LDAP authentication is enabled by the -l option. (Specify the short username, not the full LDAP distinguished name.) The shell then prompts interactively for the password.
--ldap_password_cmd= *COMMAND*	N/A	Specifies a command to run to retrieve the LDAP password, when LDAP authentication is enabled by the -l option. If the command includes space-separated arguments, enclose the command and its arguments in quotation marks.
-j, --jwt	N/A	Indicates that JWT authentication will be used when this command line option is specified in the impala-shell command.
--jwt_cmd	N/A	Shell command to run to retrieve the JWT to be used for authentication.
--config_file= *PATH_TO_CONFIG_FILE*	N/A	Specifies the path of the file containing `impala-shell` configuration settings. The default is /etc/impalarc. This setting can only be specified on the command line.
--live_progress	N/A	Prints a progress bar showing roughly the percentage complete for each query. The information is updated interactively as the query progresses.
--live_summary	N/A	Prints a detailed report, similar to the SUMMARY command, showing progress details for each phase of query execution. The information is updated interactively as the query progresses.
--var= *VARIABLE_NAME*= *VALUE*	N/A	Defines a substitution variable that can be used within the `impala-shell` session. The variable can be substituted into statements processed by the -q or -f options, or in an interactive shell session. Within a SQL statement, you substitute the value by using the notation ${var:*VARIABLE_NAME*}.
--auth_creds_ok_in_clear	N/A	Allows LDAP authentication to be used with an insecure connection to the shell. WARNING: This will allow authentication credentials to be sent unencrypted, and hence may be vulnerable to an attack.

Command-Line Option	Configuration File Setting	Explanation
--protocol= *PROTOCOL*	N/A	Protocol to use for the connection to Impala. Valid *PROTOCOL* values are: • 'hs2': Impala-shell uses the binary TCP based transport to speak to the Impala Daemon via the HiveServer2 protocol. This is the current default setting. • 'hs2-http': Impala-shell uses HTTP transport to speak to the Impala Daemon via the HiveServer2 protocol. • 'beeswax': Impala-shell uses the binary TCP based transport to speak to the Impala Daemon via Beeswax. You cannot connect to the 3.2 or earlier versions of Impala using the 'hs2' or 'hs2-http' option. Beeswax support is deprecated and will be removed in the future.
--beeswax_compat_num_rows	beeswax_compat_num_rows=true	If set to true, this option forces impala-shell to print the line Fetched 0 row(s) in Ys for all Impala protocol statements (e.g., `REFRESH` or `INVALIDATE METADATA`), regardless of whether they return result metadata. This behavior is similar to the older Beeswax protocol and can be useful for measuring the elapsed time of metadata-only operations. Defaults to false. The one exception is the USE statement, which remains silent.

Impala Shell Configuration File

You can store a set of default settings for `impala-shell` in the `impala-shell` configuration file.

The global `impala-shell` configuration file is located in /etc/impalarc.

The user-level `impala-shell` configuration file is located in ~/.impalarc.

Note that the global-level file name is different from the user-level file name. The global-level file name does not include a dot (.) in the file name.

The default path of the global configuration file can be changed by setting the $IMPALA_SHELL_GLOBAL_CONFIG_FILE environment variable.

To specify a different file name or path for the user-level configuration file, start `impala-shell` with the --config_file `impala-shell` option set to the path of the configuration file.

Typically, an administrator creates the global configuration file for the `impala-shell`, and if the user-level configuration file exists, the options set in the user configuration file take precedence over those in the global configuration file.

In turn, any options you specify on the `impala-shell` command line override any corresponding options within the configuration file.

The `impala-shell` configuration file (global or user) must contain a header label [impala], followed by the options specific to `impala-shell`.

The `impala-shell` configuration file consists of key-value pairs, one option per line. Everything after the # character on a line is treated as a comment and ignored.

The names of the options in the configuration file are similar (although not necessarily identical) to the long-form command-line arguments to the `impala-shell` command. For the supported options in the configuration file, see Impala Shell Configuration Options on page 6.

You can specify key-value pair options using keyval, similar to the --var command-line option. For example, keyval=*VARIABLE1=VALUE1*.

The query options specified in the [impala] section override the options specified in the [impala.query_options] section.

The following example shows a configuration file that you might use during benchmarking tests. It sets verbose mode, so that the output from each SQL query is followed by timing information. `impala-shell` starts inside

the database containing the tables with the benchmark data, avoiding the need to issue a USE statement or use fully qualified table names.

In this example, the query output is formatted as delimited text rather than enclosed in ASCII art boxes, and is stored in a file rather than printed to the screen. Those options are appropriate for benchmark situations, so that the overhead of `impala-shell` formatting and printing the result set does not factor into the timing measurements. It also enables the show_profiles option. That option prints detailed performance information after each query, which might be valuable in understanding the performance of benchmark queries.

```
[impala]
verbose=true
default_db=tpc_benchmarking
write_delimited=true
output_delimiter=,
output_file=/home/tester1/benchmark_results.csv
show_profiles=true
keyval=msg1=hello,keyval=msg2=world
```

The following example shows a configuration file that connects to a specific remote Impala node, runs a single query within a particular database, then exits. Any query options predefined under the [impala.query_options] section in the configuration file take effect during the session.

You would typically use this kind of single-purpose configuration setting with the `impala-shell` command-line option --config_file=*PATH_TO_CONFIG_FILE*, to easily select between many predefined queries that could be run against different databases, hosts, or even different clusters. To run a sequence of statements instead of a single query, specify the configuration option query_file=*PATH_TO_QUERY_FILE* instead.

```
[impala]
impalad=impala-test-node1.example.com
default_db=site_stats
# Issue a predefined query and immediately exit.
query=select count(*) from web_traffic where event_date = trunc(now(),'dd')

[impala.query_options]
mem_limit=32g
```

Connecting to Impala Daemon in Impala Shell

In an `impala-shell` session, you need to connect to an `impalad` daemon to issue queries. When you connect to an impalad, and that daemon coordinates the execution of all queries sent to it.

About this task

Specify the connection information using the following options:

- Through command-line options when you run the `impala-shell` command.
- Through a configuration file that is read when you run the `impala-shell` command.
- During an `impala-shell` session, by issuing a CONNECT command.



Note: You cannot connect to the 3.2 or earlier versions of Impala using the 'hs2' or 'hs2-http' protocol (--protocol option).

To connect the Impala shell during shell startup:

1. Locate the hostname that is running an instance of the `impalad` daemon. If that `impalad` uses a non-default port (something other than port 21050) for `impala-shell` connections, find out the port number also.
2. Use the -i option to the `impala-shell` interpreter to specify the connection information for that instance of `impalad`:

```
# When you are connecting to an impalad running on the same machine.
# The prompt will reflect the current hostname.
```

```
$ impala-shell
# When you are connecting to an impalad running on a remote machine, and
 impalad is listening
# on a non-default port over the HTTP HiveServer2 protocol.
$ impala-shell -i SOME.OTHER.HOSTNAME:PORT_NUMBER --protocol='hs2-http'
# When you are connecting to an impalad running on a remote machine, and
impalad is listening
# on a non-default port.
$ impala-shell -i SOME.OTHER.HOSTNAME:PORT_NUMBER
```

To connect to an Impala in the`impala-shell` session:

1. Start the Impala shell with no connection:

```
impala-shell
```

2. Locate the hostname that is running the `impalad` daemon. If that `impalad` uses a non-default port (something other than port 21050) for `impala-shell` connections, find out the port number also.
3. Use the connect command to connect to an Impala instance. Enter a command and replace *IMPALAD-HOST* with the hostname you have configured to run Impala in your environment.

```
[Not connected] > connect IMPALAD-HOST
[IMPALAD-HOST:21050] >
```

To start `impala-shell` in a specific database:

You can use all the same connection options as in previous examples. For simplicity, these examples assume that you are logged into one of the Impala daemons.

1. Find the name of the database containing the relevant tables, views, and so on that you want to operate on.
2. Use the -d option to the `impala-shell` interpreter to connect and immediately switch to the specified database, without the need for a USE statement or fully qualified names:

```
# Subsequent queries with unqualified names operate on
# tables, views, and so on inside the database named 'staging'.
$ impala-shell -i localhost -d staging

# It is common during development, ETL, benchmarking, and so on
# to have different databases containing the same table names
# but with different contents or layouts.
$ impala-shell -i localhost -d parquet_snappy_compression
$ impala-shell -i localhost -d parquet_gzip_compression
```

To run one or several statements in non-interactive mode:

You can use all the same connection options as in previous examples. For simplicity, these examples assume that you are logged into one of the Impala daemons.

1. Construct a statement, or a file containing a sequence of statements, that you want to run in an automated way, without typing or copying and pasting each time.
2. Invoke `impala-shell` with the -q option to run a single statement, or the -f option to run a sequence of statements from a file. The `impala-shell` command returns immediately, without going into the interactive interpreter.

```
# A utility command that you might run while developing shell scripts
# to manipulate HDFS files.
$ impala-shell -i localhost -d database_of_interest -q 'show tables'

# A sequence of CREATE TABLE, CREATE VIEW, and similar DDL statements
# can go into a file to make the setup process repeatable.
$ impala-shell -i localhost -d database_of_interest -f recreate_tables.sql
```

Running Commands and SQL Statements in Impala Shell

This topic provides the commonly used syntax and shortcut keys in impala-shell.

The following are a few of the key syntax and usage rules for running commands and SQL statements in impala-shell.

- To see the full set of available commands, press TAB twice.
- To cycle through and edit previous commands, click the up-arrow and down-arrow keys.
- Use the standard set of keyboard shortcuts in GNU Readline library for editing and cursor movement, such as Ctrl-A for the beginning of line and Ctrl-E for the end of line.
- Commands and SQL statements must be terminated by a semi-colon.
- Commands and SQL statements can span multiple lines.
- Use -- to denote a single-line comment and /* */ to denote a multi-line comment.

  A comment is considered part of the statement it precedes, so when you enter a -- or /* */ comment, you get a continuation prompt until you finish entering a statement ending with a semicolon. For example:

```
[impala] > -- This is a test comment
                 > SHOW TABLES LIKE 't*';
```

- If a comment contains the ${*VARIABLE_NAME*} and it is not for a variable substitution, the $ character must be escaped, e.g. -- \${hello}.

Variable Substitution in impala-shell

You can define substitution variables to be used within SQL statements processed by `impala-shell`.

1. You specify the variable and its value as below.
   - On the command line, you specify the option --var=*VARIABLE_NAME*=*VALUE*
   - Within an interactive session or a script file processed by the -f option, use the SET VAR:*VARIABLE_NAME*=*VALUE* command.
2. Use the above variable in SQL statements in the impala-shell session using the notation: ${VA R:*VARIABLE_NAME*}.

For example, here are some `impala-shell` commands that define substitution variables and then use them in SQL statements executed through the -q and -f options. Notice how the -q argument strings are single-quoted to prevent shell expansion of the ${var:value} notation, and any string literals within the queries are enclosed by double quotation marks.

```
$ impala-shell --var=tname=table1 --var=colname=x --var=coltype=string -q '
CREATE TABLE ${var:tname} (${var:colname} ${var:coltype}) STORED AS PARQUET'
Query: CREATE TABLE table1 (x STRING) STORED AS PARQUET
```

The below example shows a substitution variable passed in by the --var option, and then referenced by statements issued interactively. Then the variable is reset with the SET command.

```
$ impala-shell --quiet --var=tname=table1

[impala] > SELECT COUNT(*) FROM ${var:tname};

[impala] > SET VAR:tname=table2;
[impala] > SELECT COUNT(*) FROM ${var:tname};
```

When you run a query, the live progress bar appears in the output of a query. The bar shows roughly the percentage of completed processing. When the query finishes, the live progress bar disappears from the console output.

Impala Shell Command Reference

Use the following commands within impala-shell to pass requests to the impalad daemon that the shell is connected to. You can enter a command interactively at the prompt or pass it as the argument to the -q option of impala-shell.

Command	Explanation
Impala SQL statements	You can issue valid SQL statements to be executed.
connect	Connects to the specified instance of impalad. The default port of 21050 is assumed unless you provide another value. You can connect to any host in your cluster that is running impalad. If you connect to an instance of impalad that was started with an alternate port specified by the --fe_port flag, you must provide that alternate port.
help	Help provides a list of all available commands and options.
history	Maintains an enumerated cross-session command history. This history is stored in the ~/.impalahistory file.
profile	Displays low-level information about the most recent query. Used for performance diagnosis and tuning. The report starts with the same information as produced by the EXPLAIN statement and the SUMMARY command.
quit	Exits the shell. Remember to include the final semicolon so that the shell recognizes the end of the command.
rerun or @	Executes a previous `impala-shell` command again, from the list of commands displayed by the history command. These could be SQL statements, or commands specific to `impala-shell` such as quit or profile. Specify an integer argument. A positive integer N represents the command labelled N in the output of the HIST ORY command. A negative integer -N represents the Nth command from the end of the list, such as -1 for the most recent command. Commands that are executed again do not produce new entries in the HISTORY output list.
set	Manages query options for an `impala-shell` session. These options are used for query tuning and troubleshooting. Issue SET with no arguments to see the current query options, either based on the `impalad` defaults, as specified by you at `impalad` startup, or based on earlier SET statements in the same session. To modify option values, issue commands with the syntax set *OPTION=VALUE*. To restore an option to its default, use the unset command.
shell	Executes the specified command in the operating system shell without exiting impala-shell. You can use the ! character as shorthand for the shell command.  **Note:** Quote any instances of the -- or /* tokens to avoid them being interpreted as the start of a comment. To embed comments within source or ! commands, use the shell comment character # before the comment portion of the line.
source or src	Executes one or more statements residing in a specified file from the local filesystem. Allows you to perform the same kinds of batch operations as with the -f option, but interactively within the interpreter. The file can contain SQL statements and other `impala-shell` commands, including additional SOURCE commands to perform a flexible sequence of actions. Each command or statement, except the last one in the file, must end with a semicolon.
summary	Summarizes the work performed in various stages of a query. It provides a higher-level view of the information displayed by the EXPLAIN command. Added in Impala 1.4.0. The time, memory usage, and so on reported by SUMMARY only include the portions of the statement that read data, not when data is written. Therefore, the PROFILE command is better for checking the performance and scalability of INSERT statements. You can see a continuously updated report of the summary information while a query is in progress.
unset	Removes any user-specified value for a query option and returns the option to its default value. You can also use it to remove user-specified substitution variables using the notation UNSET VA R:*VARIABLE_NAME*.
use	Indicates the database against which to run subsequent commands. Lets you avoid using fully qualified names when referring to tables in databases other than default. Not effective with the -q option, because that option only allows a single statement in the argument.
version	Returns Impala version information.

Configuring ODBC for Impala

Download and configure the ODBC driver to integrate your applications with Impala.

About this task

Impala has been tested with the Impala ODBC driver version 2.5.42, and Cloudera recommends that you use this version with the current version of Impala.

Procedure

1. Download and install an ODBC driver.
2. Configure the ODBC port.

   Versions 2.5 and 2.0 of the Cloudera ODBC Connector use the HiveServer2 protocol, corresponding to Impala port 21050.

   Version 1.x of the Cloudera ODBC Connector uses the original HiveServer1 protocol, corresponding to Impala port 21000.

Configuring JDBC for Impala

Download and configure the JDBC driver to access Impala from a Java program that you write, or a Business Intelligence or similar tool that uses JDBC to communicate with database products.

About this task

The following are the default ports that Impala server accepts JDBC connections through:

Protocol	Default Port	Flag to Specify an Alternate Port
HTTP	28000	##hs2_http_port
Binary TCP	21050	##hs2_port

Make sure the port for the protocol you are using is available for communication with clients, for example, that it is not blocked by firewall software.

If your JDBC client software connects to a different port, specify that alternative port number with the flag in the above table when starting the impalad.

Procedure

1. Configure the JDBC port.

   Impala server accepts JDBC connections through port 21050 by default. Make sure this port is available for communication with other hosts on your network, for example, that it is not blocked by firewall software. If your JDBC client software connects to a different port, specify that alternative port number in the Impala Daemon HiveServer2 Portfield in Cloudera Manager, in the Configuration tab.
2. Install the JDBC driver.

   Impala has been tested using the Impala JDBC driver version 2.5.45 and 2.6.2. Cloudera recommends that you use one of these two versions with Impala.
3. Enable Impala JDBC support on client systems.
4. Establish JDBC connections.

   The JDBC driver class depends on which driver you select.

   Using the Cloudera JDBC Connector (recommended):

Depending on the level of the JDBC API your application is targeting, you can use the following fully-qualified class names (FQCNs):

- com.cloudera.impala.jdbc41.Driver
- com.cloudera.impala.jdbc41.DataSource
- com.cloudera.impala.jdbc4.Driver
- com.cloudera.impala.jdbc4.DataSource
- com.cloudera.impala.jdbc3.Driver
- com.cloudera.impala.jdbc3.DataSource

The connection string has the following format:

```
jdbc:impala://HOST:PORT[/SCHEMA];PROPERTY1=VALUE;PROPERTY2=VALUE;...
```

The port value is typically 21050 for Impala.

To connect to an instance of Impala that requires Kerberos authentication, use a connection string of the form jdbc :impala://*HOST*:*PORT*/;principal=*PRINCIPAL_NAME*. The principal must be the same user principal you used when starting Impala.

To connect to an instance of Impala that requires LDAP authentication, use a connection string of the form jdbc :impala://*HOST*:*PORT*/*DB_NAME*;user=*LDAP_USERID*;password=*LDAP_PASSWORD*.

To connect to an instance of Impala over HTTP, specify the HTTP port, 28000 by default, and transportMode=ht tp and httpPath=cliservice in the connection string.

Note: To establish a connection with an Impala instance from your client, using any authentication mode, you must use the connection string jdbc:impala. Using the connection string jdbc:hive2 is not recommended and is not supported.

For updated information on the version of the JDBC driver you are using and for connection string examples for different supported authentications, refer to the link provided under Related Information.

Configuring Impyla for Impala

Explains how to install Impyla to connect to and submit SQL queries to Impala. Impyla is a Python client wrapper around the HiveServer2 Thrift Service. It connects to Impala and implements Python DB API 2.0.

About this task

Impyla releases are available at pypi.org. To get the available releases, check Release history.

Note: Cloudera will not support versions of Impyla that are built manually from source code.

Key Features of Impyla

- HiveServer2 compliant.
- Works with Impala including nested data.
- DB API 2.0 (PEP 249)-compliant Python client (similar to sqlite or MySQL clients) supporting Python 2.6+ and Python 3.3+.
- Works with Kerberos, LDAP, SSL.
- SQLAlchemy connector.
- Converts to pandas DataFrame, allowing easy integration into the Python data stack (including scikit-learn and matplotlib); see the Ibis project for a richer experience.
- For more information, see here.

Before you begin

Different systems require different packages to be installed to enable SASL support in Impyla. The following list shows some examples of how to install the packages on different distributions.

You must have the following installed in your environment before installing impyla. Python 2.6+ or 3.3+ and the pip packages six, bitarray, thrift and thriftpy2 will be automatically installed as dependencies when installing impyla. However if you clone the impyla repo and run their local copy, you must install these pip packages manually.

- Install the latest pip and setuptools:

```
python -m pip install --upgrade pip setuptools
```

  Optionally, to install Impyla with Hive and/or GSSAPI (kerberos) support, you will also need to install additional software packages:
- RHEL/CentOS:

```
sudo yum install gcc-c++ cyrus-sasl-md5 cyrus-sasl-plain cyrus-sasl-gssapi
 cyrus-sasl-devel
```

- Ubuntu:

```
sudo apt install g++ libsasl2-dev libsasl2-2 libsasl2-modules-gssapi-mit
```

Procedure

1. Using pip you can install the latest release: pip install impyla
2. Optionally, to install Impyla with GSSAPI (kerberos) support:pip install impyla[kerberos]
3. You also need to pip-install pandas for conversion to DataFrame objects or sqlalchemy for the SQLAlchemy engine.

Example

Sample codes

Impyla implements the Python DB API v2.0 (PEP 249) database interface (refer to it for API details):

```
from impala.dbapi import connect

conn = connect(host = "my.host.com", port = 21050)
cursor = conn.cursor()
cursor.execute("SELECT * FROM mytable LIMIT 100")
print(cursor.description)  # prints the result set's schema
results = cursor.fetchall()
cursor.close()
conn.close()
```

The Cursorobject also exposes the iterator interface, which is buffered (controlled by cursor.arraysize):

```
cursor.execute("SELECT * FROM mytable LIMIT 100")
for row in cursor:
    print(row)
```

Furthermore the Cursor object returns you information about the columns returned in the query. This is useful to export your data as a csv file.

```
import csv

cursor.execute("SELECT * FROM mytable LIMIT 100")
columns = [datum[0] for datum in cursor.description]
targetfile = "/tmp/foo.csv"
```

```
with open(targetfile, "w", newline = "") as outcsv:
    writer = csv.writer(
        outcsv,
        delimiter = ",",
        quotechar = '"',
        quoting = csv.QUOTE_ALL,
        lineterminator = "\n")
writer.writerow(columns)
for row in cursor:
    writer.writerow(row)
```

You can also get back a pandas DataFrame object

```
from impala.util import as_pandas

# carry df through scikit-learn, for example
df = as_pandas(cur)
```

Connecting over HTTP/HTTPS

GSSAPI (kerberos) authentication over HTTP has been supported since 0.17a1. Use this example to establish connection over HTTP/HTTPS.

```
from impala.dbapi import connect

conn = connect(
    "impala-coordinator.example.com",
    28000,
    auth_mechanism = "GSSAPI",
    kerberos_service_name = "impala",
    use_http_transport = True,
    http_path = "cliservice",
    auth_cookie_name = "impala.auth")
cursor = conn.cursor()
cursor.execute("SHOW DATABASES")
res = cursor.fetchall()
cursor.close()
conn.close()
```

Connecting to Cloudera Data Hub

Lists an example code to connect to Impala with LDAP over http using LDAP as the authentication mechanism.

Example

Sample code

```
from impala.dbapi import connect

conn = connect(
    host = "aaaaaaa-aaaa-master0.se-sandb.a465-9q4k.cloudera.site",
    port = 443,
    auth_mechanism = "LDAP",
    use_ssl = True,
    use_http_transport = True,
    http_path = "aaaaaaa-aaaa/cdp-proxy-api/impala",
    user = "aaaaaaa",
    password = "xxxxx")
cursor = conn.cursor()
cursor.execute("SELECT * FROM default.emax_temp")

for row in cursor:
```

```
    print(row)
cursor.close()
conn.close()
```

Connecting to Cloudera Data Hub Data Mart

Lists an example code to connect to Cloudera Data Hub Data Mart using LDAP as the authentication mechanism.

Before you begin

Must have the latest Impyla release.

Example

Sample code

```
from impala.dbapi import connect

conn = connect(
    "xxxxxxx-data-mart-master0.xxxxxxx.xcu2-8y8x.dev.cldr.work",
    443,
    auth_mechanism = "LDAP",
    user = "XXXXX",
    password = "XXXXX",
    use_ssl = True,
    use_http_transport = True,
    http_path = "xxxxxxx-data-mart/cdp-proxy-api/impala")
cursor = conn.cursor()
cursor.execute("SHOW DATABASES")
res = cursor.fetchall()
print(res)
cursor.close()
conn.close()
```

Configuring Delegation for Clients

Impala supports user and group delegation for client connections.

About this task

When users submit Impala queries through a separate application, such as Data Explorer or a business intelligence tool, typically all requests are treated as coming from the same user. Impala supports “delegation” where users whose names you specify can delegate the execution of a query to another user. The query runs with the privileges of the delegated user, not the original authenticated user.

You also have an option to delegate using groups. Instead of listing a large number of delegated users, you can create a group of those users and specify the delegated group name in the Impalad startup option. The client sends the delegated user name, and Impala performs an authorization to see if the delegated user belongs to a delegated group.

The name of the delegated user is passed using the HiveServer2 protocol configuration property impala.doas.user when the client connects to Impala.

When the client connects over HTTP, the doAs parameter can be specified in the HTTP path. For example:

```
/?doAs=DELEGATED_USER
```

Currently, the delegation feature is available only for Impala queries submitted through application interfaces such as Data Explorer and BI tools. For example, Impala cannot issue queries using the privileges of the HDFS user.

Attention:

- When the delegation is enabled in Impala, the Impala clients should take an extra caution to prevent unauthorized access for the delegate-able users.
- Impala requires Apache Ranger on the cluster to enable delegation. Without Ranger installed, the delegation feature will fail with the following error: User user1 is not authorized to delegate to user2. User/group delegation is disabled.

Procedure

To enable delegation:

1. In Cloudera Manager, navigate to Clusters Impala .
2. In the Configuration tab, click Impala-1 (Service-Wide) in the Scope and click Security in the Category.
3. In the Proxy User Configuration field, type the a semicolon-separated list of key=value pairs of authorized proxy users to the user(s) they can impersonate.

   The list of delegated users are delimited with a comma, e.g. hue=user1, user2.
4. In the Proxy Group Configuration field, type the a semicolon-separated list of key=value pairs of authorized proxy users to the group(s) they can impersonate.

   The list of delegated groups are delimited with a comma, e.g. hue=group1, group2.
5. Click Save Changes and restart the Impala service.

Spooling Impala Query Results

In Impala, you can control how query results are materialized and returned to clients, e.g. impala-shell, Data Explorer, JDBC apps.

Result spooling is turned off by default, but can be enabled via the SPOOL_QUERY_RESULTS query option.

- When query result spooling is disabled, Impala relies on clients to fetch results to trigger the generation of more result row batches until all the result rows have been produced. If a client issues a query without fetching all the results then that query continues to be in the running state indefinitely and the query fragments continue to consume the resources until the query is cancelled and unregistered, potentially tying up resources and causing other queries to wait for an extended period of time in admission control.

  Impala would materialize rows on-demand where rows are created only when the client requests them.

  For example, if a Data Explorer user runs a complex query that returns 1000 rows, but does not scroll through all the returned rows, and then stays idle for a while, the query will remain running and will hold onto all of its resources until it is explicitly closed or the session times out.
- When query result spooling is enabled, result sets of queries are eagerly fetched and spooled in the spooling location, either in memory or on disk.

  Once all result rows have been fetched and stored in the spooling location, the resources are freed up. Incoming client fetches can get the data from the spooled results.

Admission Control and Result Spooling

Query results spooling collects and stores query results in memory that is controlled by admission control. Use the following query options to calibrate how much memory to use and when to spill to disk.

MAX_RESULT_SPOOLING_MEM

The maximum amount of memory used when spooling query results. If this value is exceeded when spooling results, all memory will most likely be spilled to disk. Set to 100 MB by default.

MAX_SPILLED_RESULT_SPOOLING_MEM

The maximum amount of memory that can be spilled to disk when spooling query results. Must be greater than or equal to MAX_RESULT_SPOOLING_MEM. If this value is exceeded, the

coordinator fragment will block until the client has consumed enough rows to free up more memory. Set to 1 GB by default.

Fetch Timeout

Resources for a query are released when the query completes its execution. To prevent clients from indefinitely waiting for query results, use the FETCH_ROWS_TIMEOUT_MS query option to set the timeout when clients fetch rows. Timeout applies both when query result spooling is enabled and disabled:

- When result spooling is disabled (SPOOL_QUERY_RESULTS = FALSE), the timeout controls how long a client waits for a single row batch to be produced by the coordinator.
- When result spooling is enabled ((SPOOL_QUERY_RESULTS = TRUE), a client can fetch multiple row batches at a time, so this timeout controls the total time a client waits for row batches to be produced.

Explain Plans

Below is the part of the EXPLAIN plan output for result spooling.

```
F01:PLAN FRAGMENT [UNPARTITIONED] hosts=1 instances=1
|  Per-Host Resources: mem-estimate=4.02MB mem-reservation=4.00MB thread-r
eservation=1
PLAN-ROOT SINK
|  mem-estimate=4.00MB mem-reservation=4.00MB spill-buffer=2.00MB thread-res
ervation=0
```

- The mem-estimate for the PLAN-ROOT SINK is an estimate of the amount of memory needed to spool all the rows returned by the query.
- The mem-reservation is the number and size of the buffers necessary to spool the query results. By default, the read and write buffers are 2 MB in size each, which is why the default is 4 MB.

PlanRootSink

In Impala, the PlanRootSink class controls the passing of batches of rows to the clients and acts as a queue of rows to be sent to clients.

- When result spooling is disabled, a single batch or rows is sent to the PlanRootSink, and then the client must consume that batch before another one can be sent.
- When result spooling is enabled, multiple batches of rows can be sent to the PlanRootSink, and multiple batches can be consumed by the client.

Configuring Impala

You must review and configure the mandatory and recommended settings if you installed Impala without Cloudera Manager or if you want to customize your environment after installing Impala. If you installed Impala using Cloudera Manager, some of these configurations are completed automatically.

In some cases, depending on the level of Impala, Cloudera, and Cloudera Manager, you might need to add particular component configuration details in one of the free-form fields on the Impala configuration pages in Cloudera Manager.

- You must enable short-circuit reads, whether or not Impala was installed through Cloudera Manager. This setting goes in the Impala configuration settings, not the Hadoop-wide settings.
- If you installed Impala in an environment that is not managed by Cloudera Manager, you must enable block location tracking, and you can optionally enable native checksumming for optimal performance.

Short-Circuit Reads

Enabling short-circuit reads allows Impala to read local data directly from the file system. This removes the need to communicate through the DataNodes, improving performance. This setting also minimizes the number of additional copies of data. Short-circuit reads requires libhadoop.so (the Hadoop Native Library) to be accessible to both the server and the client. You must install it from an .rpm, .deb, or parcel to use short-circuit local reads.

Note: If you use Cloudera Manager, you can enable short-circuit reads through a checkbox in the user interface and that setting takes effect for Impala as well.

To Enable Short-Circuit Reads

You can enable short-circuit reads through a checkbox available under Configuration tab for both Impala and HDFS.

1. In Cloudera Manager Clusters select the Impala service, for example, IMPALA.
2. On the Configuration tab, search for dfs.client.read.shortcircuit.
3. Accept the default (enabled), or check to enable the dfs.client.read.shortcircuit property to read the HDFS file blocks directly.
4. Do the above for HDFS service too by clicking Cloudera Manager Clusters and by selecting the HDFS service.
5. Save the changes and Restart the service.

To configure DataNodes for short-circuit reads:

1. On all Impala nodes, configure the following attributes from the HDFS service as shown:
   a. In Cloudera Manager Clusters select the HDFS service, for example, HDFS.
   b. On the Configuration tab, search for dfs.domain.socket.path and set the attribute.
   c. Search for and set the attribute, if necessary, dfs.client.file-block-storage-locations.timeout.millis.
   d. Search for and set the attribute, if necessary, dfs.datanode.hdfs-blocks-metadata.enabled
2. After applying these changes, restart all DataNodes.

Block Location Tracking

Enabling block location metadata allows Impala to know which disk data blocks are located on, allowing better utilization of the underlying disks. Impala will not start unless this setting is enabled.

To enable block location tracking:

1. For each DataNode, enable the following attribute dfs.datanode.hdfs-blocks-metadata.enabled file:
   a. In Cloudera Manager Clusters select the HDFS service, for example, HDFS.
   b. On the Configuration tab, search for dfs.datanode.hdfs-blocks-metadata.enabled and enable the attribute if not already enabled.
2. After applying these changes, restart all DataNodes.

Native Checksumming

Enabling native checksumming causes Impala to use an optimized native library for computing checksums, if that library is available.

To enable native checksumming:

If you installed Cloudera from packages, the native checksumming library is installed and setup correctly, and no additional steps are required.

If you installed by other means, native checksumming may not be available due to missing shared objects. Finding the message "Unable to load native-hadoop library for your platform... using builtin-java classes where applicable" in the Impala logs indicates native checksumming may be unavailable.

To enable native checksumming, you must build and install libhadoop.so (the Hadoop Native Library).

Configuring Impala for High Availability

You can learn how to ensure high availability in an Impala cluster by deploying pairs of StateStore and Catalog instances in primary/standby mode. This setup reduces single points of failure and minimizes outage durations by allowing standby instances to take over when primary instances fail.

How configuring High Availability helps

The Impala StateStore checks on the health of all Impala daemons in a cluster, and continuously relays its findings to each of the daemons. The Catalog stores metadata of databases, tables, partitions, resource usage information, configuration settings, and other objects managed by Impala. If StateStore and Catalog daemons are single instances in an Impala cluster, it creates a single point of failure. Although Impala coordinators/executors continue to execute queries if the StateStore node is down, coordinators/executors will not get state updates. This causes degradation of admission control and cluster membership updates. To mitigate this, a pair of StateStore and Catalog instances can be deployed in an Impala cluster so that Impala cluster can survive failures of StateStore or Catalog.

How High Availability works in Impala cluster

With a pair of StateStore instances in primary/standby mode, the primary StateStore instance will send the cluster's state update and propagate metadata updates. It periodically sends heartbeat to the standby StateStore instance, Catalog, coordinators and executors. The standby StateStore instance also sends heartbeats to the Catalog, and coordinators and executors. RPC connections between daemons and StateStore instances are kept alive so that broken connections usually do not result in false failure reports between nodes. The standby StateStore instance takes over the primary role when the service is needed in order to continue to operate when the primary instance goes down.

With any new query requests, the Impala coordinator sends metadata requests to catalog service and sends metadata updates to catalog which in turn propagates metadata updates to hive metastore. With a pair of primary/standby Catalog instances, the standby catalog instance is promoted as the primary instance to continue Catalog service for Impala cluster when the primary instance goes down. The active catalogd acts as the source of metadata and provides Catalog service for the Impala cluster. This High Availability (HA) mode of Catalog service reduces the outage duration of the Impala cluster when the primary catalog instance fails. To support Catalog HA, you can now add two catalogd instances in an Active-Passive HA pair to an Impala cluster by choosing the High Availability option.

Enabling Catalog and StateStore High Availability (HA)

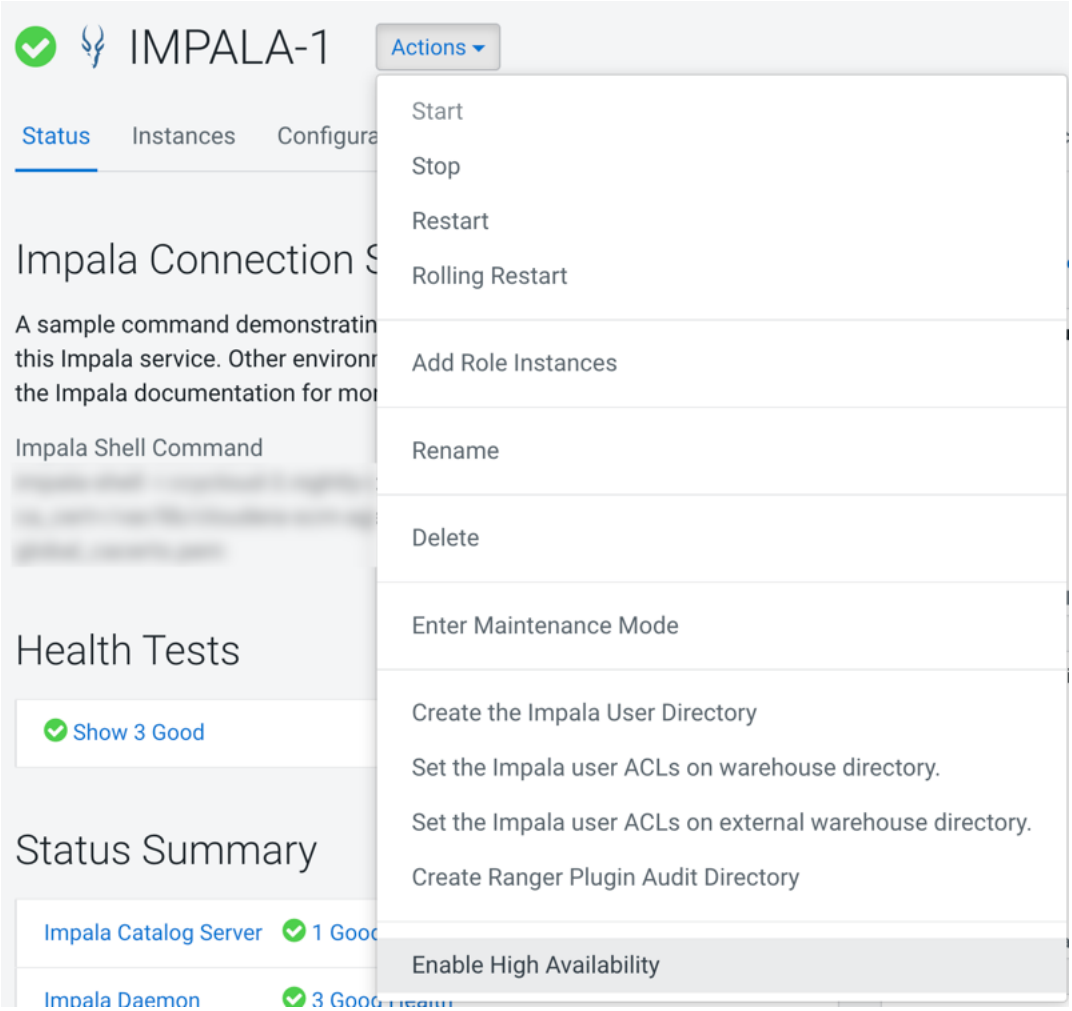
You can use Cloudera Managerto enable Catalog and StateStore HA for a cluster. When enabled, the preemptive behavior allows one of the Catalogd and StateStore pair to become active, and the other paired Catalogd and StateStore to become a standby.

Before you begin

- Impala High Availability is supported starting with 7.1.9 SP1 and Cloudera Manager 7.11.3 Cumulative Hotfix 7.
- To avoid incompatible Impala versions, High Availability should be enabled before or after rolling upgrades.
- Ensure that all nodes running Impala service are using the same Cloudera Parcel version.

Procedure

1. Log in to Cloudera Manageras an Administrator
2. In Cloudera Manager, select Impala Actions Enable High Availability .



3. On the **Assign Roles** page click Select a host for the Catalog to act as a standby Impala Catalog server.
4. On the **Hosts Selected** page, the existing Catalog instance is selected by default. Select a backup Catalog host.

   An **ICS** icon appears in the **Added Roles** column for the Catalog server.
5. On the **Assign Roles** page click Select a host for the StateStore to act as a standby Impala StateStore server.
6. On the **Hosts Selected** page, the existing StateStore instance is selected by default. Select a backup StateStore host.

   An **ISS** icon appears in the **Added Roles** column for the StateStore server.
7. Click **OK**.
8. Click **Continue** on the **Review Changes** page.

   Review the summary of actions when you add a new host. When a new host is selected, Impala and its dependent services are restarted.

   Important:

   During a Catalogd restart, metadata operations including creating new tables, adding new partitions, and invalidating metadata do not work until the Catalog service is fully restarted.

Results

You notice two Impala Catalog servers and two Impala StateStore servers under Impala > Instances page.

Disabling Catalog and StateStore High Availability

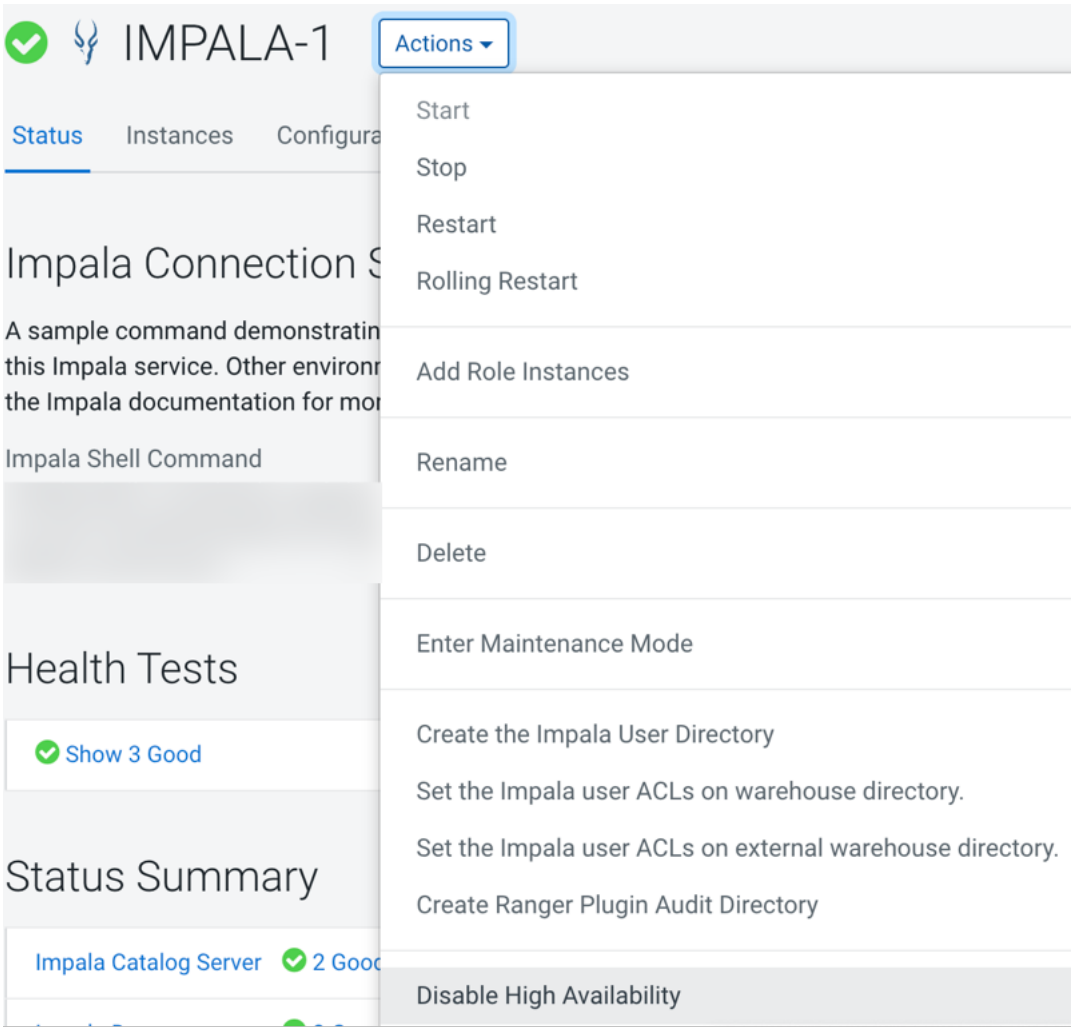
You can use Cloudera Manager to disable Catalog and StateStore High Availability (HA) for a cluster.

Before you begin

Ensure that all nodes running Impala service are using the same Cloudera Parcel version.

Procedure

1. Log in to Cloudera Manager as an Administrator.
2. In Cloudera Manager, select Impala Actions Enable High Availability .



3. On the **Getting started** page, select the hosts you want to remain active when reverting Impala Catalog and Impala StateStore to a single node configuration. You will need to select one host for the Impala Catalog and one host for the Impala StateStore.
4. Click Continue.

   All the dependent services are restarted.

Results

You will notice one Impala Catalog server and one Impala StateStore server under Impala Instances page.

Failure detection for Catalog and StateStore

Learn how the Impala High Availability (HA) system maintains continuous operation by switching from primary to standby instances when a heartbeat check fails.

Catalog Failure Detection

The StateStore instance continuously sends heartbeat to its registered clients, including the primary and standby Catalog instances, to track Impala daemons in the cluster to determine if the daemon is healthy. If the StateStore finds the primary Catalog instance is not healthy but the standby Catalog instance is healthy, StateStore promotes the standby Catalog instance as primary instance and notifies all coordinators about this change. Coordinators switch over to the new primary Catalog instance.

When the system detects that the active catalogd is unhealthy, it initiates a failover to the standby Catalogd. During this brief transition, some nodes might not immediately recognize the new active Catalogd, causing currently running queries to fail due to lack of access to metadata. These failed queries need to be rerun after the failover is complete and the new active Catalogd is operational.

StateStore Failure Detection

The primary StateStore instance continuously sends heartbeat to its registered clients, and the standby StateStore instance. Each StateStore client registers with both active and standby statestore instances, and maintains the following information about the StateStore servers: the server IP and port, service role - primary/standby, the last time the heartbeat request was received, or number of missed heartbeats. A missing heartbeat response from the StateStore's client indicates an unhealthy daemon. There is a flag that defines MAX_MISSED_HEARTBEAT_REQUEST_NUM as the consecutive number of missed heartbeat requests to indicate losing communication with the StateStore server from the client's point of view so that the client marks the StateStore server as down. Standby StateStore instance collects the connection states between the clients (Catalog, coordinators and executors) and primary StateStore instance in its heartbeat messages to the clients. If the standby StateStore instance misses MAX_MISSED_HEARTBEAT_REQUEST_NUM of heartbeat requests from the primary StateStore instance and majority of clients lose connections with the primary statestore, it takes over the primary role.

Configuring Load Balancer for Impala

For most clusters that have multiple users and production availability requirements, you might want to set up a load-balancing proxy server to relay requests to and from Impala.

When using a load balancer for Impala, applications connect to a single well-known host and port, rather than keeping track of the hosts where a specific Impala daemon is running. The load balancer also lets the Impala nodes share resources to balance out the work loads.

Set up a software package of your choice to perform these functions.

Most considerations for load balancing and high availability apply to the impalad daemons. The statestored and catalogd daemons do not have special requirements for high availability, because problems with those daemons do not result in data loss.

The following are the general setup steps that apply to any load-balancing proxy software:

1. Select and download a load-balancing proxy software or other load-balancing hardware appliance. It should only need to be installed and configured on a single host, typically on an edge node.
2. Configure the load balancer (typically by editing a configuration file). In particular:
   - To relay Impala requests back and forth, set up a port that the load balancer will listen on.
   - Select a load balancing algorithm.
   - For Kerberized clusters, follow the instructions in Special Proxy Considerations for Clusters Using Kerberos on page 26 below.

3. If you are using Data Explorer or JDBC-based applications, you typically set up load balancing for both ports 21000 and 21050, because these client applications connect through port 21050 while the `impala-shell` command connects through port 21000. See Ports used by Impala for when to use port 21000, 21050, or another value depending on what type of connections you are load balancing.
4. Run the load-balancing proxy server, pointing it at the configuration file that you set up.
5. In Cloudera Manager, navigate to Impala Configuration Impala Daemon Default Group .
6. In the Impala Daemons Load Balancer field, specify the address of the load balancer in the *HOST*:*PORT* format.

   This setting lets Cloudera Manager route all appropriate Impala-related operations through the load-balancing proxy server.
7. For any scripts, jobs, or configuration settings for applications that formerly connected to a specific `impalad` to run Impala SQL statements, change the connection information (such as the -i option in `impala-shell`) to point to the load balancer instead.



Note: The following sections use the HAProxy software as a representative example of a load balancer that you can use with Impala.

Choosing the Load-Balancing Algorithm

Load-balancing software offers a number of algorithms to distribute requests. Each algorithm has its own characteristics that make it suitable in some situations but not others.

Leastconn

Connects sessions to the coordinator with the fewest connections, to balance the load evenly. Typically used for workloads consisting of many independent, short-running queries. In configurations with only a few client machines, this setting can avoid having all requests go to only a small set of coordinators.

Recommended for Impala with F5.

Source IP Persistence

Sessions from the same IP address always go to the same coordinator. A good choice for Impala workloads containing a mix of queries and DDL statements, such as CREATE TABLE and ALTE R TABLE. Because the metadata changes from a DDL statement take time to propagate across the cluster, prefer to use the Source IP Persistence algorithm in this case. If you are unable to choose Source IP Persistence, run the DDL and subsequent queries that depend on the results of the DDL through the same session, for example by running impala-shell -f *SCRIPT_FILE* to submit several statements through a single session.

Required for setting up high availability with Data Explorer.

Round-robin

Distributes connections to all coordinator nodes. Typically not recommended for Impala.

You might need to perform benchmarks and load testing to determine which setting is optimal for your use case. Always set up using two load-balancing algorithms: Source IP Persistence for Data Explorer and Leastconn for others.

Special Proxy Considerations for Clusters Using Kerberos

In a cluster using Kerberos, applications check host credentials to verify that the host they are connecting to is the same one that is actually processing the request.

In Impala 2.11 and lower versions, once you enable a proxy server in a Kerberized cluster, users will not be able to connect to individual impala daemons directly from impala-shell.

In Impala 2.12 and higher, when you enable a proxy server in a Kerberized cluster, users have an option to connect to Impala daemons directly from `impala-shell` using the -b / --kerberos_host_fqdn `impala-shell` flag. This option can be used for testing or troubleshooting purposes, but not recommended for live production environments as it defeats the purpose of a load balancer/proxy.

Example:

```
impala-shell -i impalad-1.mydomain.com -k -b loadbalancer-1.mydomain.com
```

Alternatively, with the fully qualified configurations:

```
impala-shell --impalad=impalad-1.mydomain.com:21000 --kerberos --kerberos_ho
st_fqdn=loadbalancer-1.mydomain.com
```

To validate the load-balancing proxy server, perform these extra Kerberos setup steps:

1. This section assumes you are starting with a Kerberos-enabled cluster.
2. Choose the host you will use for the proxy server. Based on the Kerberos setup procedure, it should already have an entry impala/*PROXY_HOST@REALM* in its keytab.
3. In Cloudera Manager, navigate to Impala Configuration Impala Daemon Default Group .
4. In the Impala Daemons Load Balancer field, specify the address of the load balancer in the *HOST*:*PORT* format.

   When this field is specified and Kerberos is enabled, Cloudera Manager adds a principal for impa la/*PROXY_HOST@REALM* to the keytab for all Impala daemons.
5. Restart the Impala service.

Client Connection to Proxy Server in Kerberized Clusters

When a client connect to Impala, the service principal specified by the client must match the -principal setting, impa la/*PROXY_HOST@REALM*, of the Impala proxy server as specified in its keytab. And the client should connect to the proxy server port.

In hue.ini, set the following to configure Data Explorer to automatically connect to the proxy server:

```
[impala]
server_host=PROXY_HOST
impala_principal=impala/PROXY_HOST
```

The following are the JDBC connection string formats when connecting through the load balancer with the load balancer's host name in the principal:

```
jdbc:hive2://PROXY_HOST:LOAD_BALANCER_PORT/;principal=impala/_HOST@REALM
jdbc:hive2://PROXY_HOST:LOAD_BALANCER_PORT/;principal=impa
la/PROXY_HOST@REALM
```

When starting `impala-shell`, specify the service principal via the -b or --kerberos_host_fqdn flag.

Special Proxy Considerations for TLS/SSL Enabled Clusters

When TLS/SSL is enabled for Impala, the client application, whether impala-shell, Data Explorer, or something else, expects the certificate common name (CN) to match the hostname that it is connected to. With no load balancing proxy server, the hostname and certificate CN are both that of the impalad instance. However, with a proxy server, the certificate presented by the impalad instance does not match the load balancing proxy server hostname. If you try to load-balance a TLS/SSL-enabled Impala installation without additional configuration, you see a certificate mismatch error when a client attempts to connect to the load balancing proxy host.

You can configure a proxy server in several ways to load balance TLS/SSL enabled Impala:

TLS/SSL Bridging

In this configuration, the proxy server presents a TLS/SSL certificate to the client, decrypts the client request, then re-encrypts the request before sending it to the backend impalad. The client and server certificates can be managed separately. The request or resulting payload is encrypted in transit at all times.

TLS/SSL Passthrough

In this configuration, traffic passes through to the backend impalad instance with no interaction from the load balancing proxy server. Traffic is still encrypted end-to-end.

The same server certificate, utilizing either wildcard or Subject Alternate Name (SAN), must be installed on each impalad instance.

TLS/SSL Offload

In this configuration, all traffic is decrypted on the load balancing proxy server, and traffic between the backend impalad instances is unencrypted. This configuration presumes that cluster hosts reside on a trusted network and only external client-facing communication need to be encrypted in-transit.

If you plan to use Auto-TLS, your load balancer must perform TLS/SSL bridging or TLS/SSL offload.

Refer to your load balancer documentation for the steps to set up Impala and the load balancer using one of the options above.

Example of Configuring HAProxy Load Balancer for Impala

If you are not already using a load-balancing proxy, you can experiment with HAProxy a free, open source load balancer.



Attention: HAProxy is not a CDH component, and Cloudera does not provide the support for HAProxy. Refer to HAProxy for questions and support issues for HAProxy.

This example shows how you might install and configure that load balancer on a Red Hat Enterprise Linux system.

- Install the load balancer:

```
yum install haproxy
```

- Set up the configuration file: /etc/haproxy/haproxy.cfg. See the following section for a sample configuration file.
- Run the load balancer (on a single host, preferably one not running `impalad`):

```
/usr/sbin/haproxy –f /etc/haproxy/haproxy.cfg
```

- In `impala-shell`, JDBC applications, or ODBC applications, connect to the listener port of the proxy host, rather than port 21000 or 21050 on a host actually running `impalad`. The sample configuration file sets haproxy to listen on port 25003, therefore you would send all requests to *HAPROXY_HOST*:25003.

This is the sample haproxy.cfg used in this example:

```
global
    # To have these messages end up in /var/log/haproxy.log you will
    # need to:
    #
    # 1) configure syslog to accept network log events.  This is done
    #    by adding the '-r' option to the SYSLOGD_OPTIONS in
    #    /etc/sysconfig/syslog
    #
    # 2) configure local2 events to go to the /var/log/haproxy.log
    #   file. A line like the following can be added to
    #   /etc/sysconfig/syslog
    #
    #    local2.*                       /var/log/haproxy.log
    #
    log         127.0.0.1 local0
    log         127.0.0.1 local1 notice
    chroot      /var/lib/haproxy
    pidfile     /var/run/haproxy.pid
    maxconn     4000
    user        haproxy
    group       haproxy
```

```
    daemon

    # turn on stats unix socket
    #stats socket /var/lib/haproxy/stats
#---------------------------------------------------------------------
# common defaults that all the 'listen' and 'backend' sections will
# use if not designated in their block
#
# You might need to adjust timing values to prevent timeouts.
#
# The timeout values should be dependant on how you use the cluster
# and how long your queries run.
#---------------------------------------------------------------------
defaults
    mode                    http
    log                     global
    option                  httplog
    option                  dontlognull
    option http-server-close
    option forwardfor       except 127.0.0.0/8
    option                  redispatch
    retries                 3
    maxconn                 3000
    timeout connect 5000
    timeout client 3600s
    timeout server 3600s

#
# This sets up the admin page for HA Proxy at port 25002.
#
listen stats :25002
    balance
    mode http
    stats enable
    stats auth USERNAME:PASSWORD

# This is the setup for Impala. Impala client connect to load_balancer_hos
t:25003.
# HAProxy will balance connections among the list of servers listed below.
# The list of Impalad is listening at port 21000 for beeswax (impala-shell)
or original ODBC driver.
# For JDBC or ODBC version 2.x driver, use port 21050 instead of 21000.
listen impala :25003
    mode tcp
    option tcplog
    balance leastconn

    server SYMBOLIC_NAME_1 impala-host-1.example.com:21000 check
    server SYMBOLIC_NAME_2 impala-host-2.example.com:21000 check
    server SYMBOLIC_NAME_3 impala-host-3.example.com:21000 check
    server SYMBOLIC_NAME_4 impala-host-4.example.com:21000 check

# Setup for Data Explorer or other JDBC-enabled applications.
# In particular, Data Explorer requires sticky sessions.
# The application connects to load_balancer_host:21051, and HAProxy balances
# connections to the associated hosts, where Impala listens for JDBC
# requests on port 21050.
listen impalajdbc :21051
    mode tcp
    option tcplog
    balance source
    server SYMBOLIC_NAME_5 impala-host-1.example.com:21050 check
    server SYMBOLIC_NAME_6 impala-host-2.example.com:21050 check
    server SYMBOLIC_NAME_7 impala-host-3.example.com:21050 check
```

```
    server SYMBOLIC_NAME_8 impala-host-4.example.com:21050 check
```

Important: Data Explorer requires the check option at the end of each line in the above file to ensure HAProxy can detect any unreachable impalad server, and failover can be successful. Without the TCP check, you can encounter an error when the `impalad` daemon to which Data Explorer tries to connect is down.

Note: If your JDBC or ODBC application connects to Impala through a load balancer such as haproxy, be cautious about reusing the connections. If the load balancer has set up connection timeout values, either check the connection frequently so that it never sits idle longer than the load balancer timeout value, or check the connection validity before using it and create a new one if the connection has been closed.

Example of Configuring HAProxy Load Balancer for Impala with Auto-TLS

1. Create unencrypted key file. Auto TLS will not create the unencrypted file so you must create it manually.

```
        openssl rsa -in  /var/lib/cloudera-scm-agent/agent-cert/cm-auto-
host_key.pem -out /etc/haproxy/cm-auto-host_unenckey.pem
        # enter password from /var/lib/cloudera-scm-agent/agent-cert/cm-
auto-host_key.pw
```

   Note: The created cm-auto-host_cert_chain_unenckey.pem should be owned by the haproxy user with 600 or 400 privileges.

2. Combine the created unencrypted key with CA certificate:

```
cat /var/lib/cloudera-scm-agent/agent-cert/cm-auto-host_cert_chain.pem /
etc/haproxy/cm-auto-host_unenckey.pem > /etc/haproxy/cm-auto-host_cert_c
hain_unenckey.pem
```

   Note: You must reload HAProxy after combining the unencrypted key with CA certificate.

3. Configure haproxy.cfg, most often found in /etc/haproxy/haproxy.cfg. This example shown here uses non-standarad ports however you may want to configure port 21000 and 21050. And you may want to use balance source instead of balance leastconn which is needed for tools like Data Explorer to work properly.

```
frontend impala-auto-tls-thrift
    bind :21013 ssl crt /etc/haproxy/cm-auto-host_cert_chain_unenckey.pem
    mode tcp
    stats enable
    default_backend shell_autotls_backend

backend shell_autotls_backend
    mode tcp
    balance leastconn
    timeout connect 5000ms
    timeout client  3600000ms
    timeout server  3600000ms
    # Impala Nodes
    server ve1311.halxg.cloudera.com ve1311.halxg.cloudera.com:21000 chec
k ssl ca-file /var/lib/cloudera-scm-agent/agent-cert/cm-auto-global_cace
rts.pem
    server ve1312.halxg.cloudera.com ve1312.halxg.cloudera.com:21000 check
 ssl ca-file /var/lib/cloudera-scm-agent/agent-cert/cm-auto-global_cacerts
.pem

frontend impala-autotls-jdbc
    bind :21056 ssl crt /etc/haproxy/cm-auto-host_cert_chain_unenckey.pem
    mode tcp
    stats enable
```

```
    default_backend jdbc_autotls_backend

backend jdbc_autotls_backend
    mode tcp
    balance leastconn
    timeout connect 5000ms
    timeout client  3600000ms
    timeout server  3600000ms
    # Impala Nodes
    server ve1311.halxg.cloudera.com ve1311.halxg.cloudera.com:21050 check
 ssl ca-file /var/lib/cloudera-scm-agent/agent-cert/cm-auto-global_cacerts
.pem
    server ve1312.halxg.cloudera.com ve1312.halxg.cloudera.com:21050 ch
eck ssl ca-file /var/lib/cloudera-scm-agent/agent-cert/cm-auto-global_ca
certs.pem
```

Related Information

Ports used by Impala

Impala Shell Configuration Options

Configuring Kerberos Authentication

Running queries on system tables

Learn how queries against system table queries can be configured to consider only the coordinator resources that are needed.

Note: This feature is in technical preview and not recommended for use in production deployments. Cloudera recommends that you try this feature in test and development environments.

Queries against Impala system tables, such as sys.impala_query_live, experienced delays due to admission control constraints. These queries, requiring only coordinator resources, were blocked by other queries competing for executor resources.

To address this, Impala introduces an "only coordinators" request pool. This allows system table queries to run without waiting for executor resources, even when those resources are fully used. In Cloudera Runtime, queries submitted to an only coordinators request pool continue to run even if no executors are running.

Coordinator-only request pools

You can configure coordinator-only request pools in Impala by setting the option to true in the fair-scheduler.xml file. When enabled, the request pool runs queries only on coordinators. No executors are required, and all fragment instances are executed exclusively on the coordinators.

Note: Any query can be submitted to a coordinator-only request pool. If a large or non-system table query is submitted, it runs entirely on the coordinators and may lead to memory or CPU exhaustion.

To avoid resource issues, Cloudera recommends limiting access to coordinator-only request pools to a small group of users who understand the risks of running complex queries in this configuration.

Configuring the only coordinators request pool

Learn how to configure a coordinator-only request pool to optimize system table queries.

About this task

Configuring a coordinator-only request pool allows you to run queries that do not require executors, reducing resource constraints and improving performance

Before you begin

Ensure you have administrator access to modify Impala configurations.

Procedure

1. Log in to Cloudera Manageras an Administrator
2. In Cloudera Manager, go to Impala Configurations Impala Daemon Fair Scheduler Advanced Configuration Snippet (Safety Valve) .
3. Add <onlyCoordinators>true</onlyCoordinators>
4. Click **Apply Changes** button and restart Impala.

Migrating Impala Catalog to another host

You can move the Impala Catalog Server role to a new host using Cloudera Manager.

Procedure

1. Log in to Cloudera Manager
2. Stop the **Impala** service and all dependent services, including **Oozie** and **Data Explorer (Hue)**.
3. Go to the **Impala instances** page and select the **Impala Catalog Server** role type.
4. Click on Actions for Selected and choose Delete to remove the existing role. Confirm the deletion.
5. Click Add Role Instances and assign the Impala Catalog Server role to the desired host.
6. Restart the Impala service.

Results

The Impala Catalog Server role now runs on a new host.