Cloudera Runtime 7.2.11

Starting and Stopping Apache Impala

Date published: 2021-01-01
Date modified: 2021-09-06



https://docs.cloudera.com/

Legal Notice

© Cloudera Inc. 2025. All rights reserved.

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

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

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

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

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

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

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

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

Contents

Modifying Impala Startup Options..... 4

Configuring Client Access to Impala.....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..... 11
Impala Shell Command Reference..... 12
Configuring ODBC for Impala..... 13
Configuring JDBC for Impala..... 14
Configuring Impyla for Impala..... 15
Connecting to DataHub..... 17
Connecting to DataHub Data Mart..... 18
Configuring Delegation for Clients..... 18
Spooling Impala Query Results..... 19

Shut Down Impala.....21

Setting Timeouts in Impala..... 21

Setting Timeout and Retries for Thrift Connections to Backend Client..... 22
Increasing StateStore Timeout..... 22
Setting the Idle Query and Idle Session Timeouts..... 23

Modifying Impala Startup Options

You can view and edit the configuration options for the Impala daemons to customize your Impala environment, such as to specify which hosts and ports to use, to assign directories for logging,and to control resource usage and security.

Configuring Impala Startup Options

Navigate to the following page to configure the settings for all the Impala-related daemons:

ClustersImpalaConfiguration.

If the Cloudera Manager interface does not yet have a form field for a newly added option, or if you need to use special options for debugging and troubleshooting, the Advanced category page for each daemon includes one or more Safety Valve fields where you can enter option names directly.

Checking the Values of Impala Startup Options

You can check the current runtime value of all these settings through the Impala Web UI, available by default at:

- http://*IMPALA_HOSTNAME*:25000/varz for the `impalad` daemon
- http://*IMPALA_HOSTNAME*:25010/varz for the `statestored` daemon
- http://*IMPALA_HOSTNAME*:25020/varz for the `catalogd` daemon

Related Information

"Web User Interface for Debugging"

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 Hue web-based user interface
- JDBC
- ODBC
- Impyla

Impala clients can connect to any Coordinator Impala Daemon (impalad) via HiveServer2 over HTTP or over the TCP binary or via 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 64 with which 64 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.

--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.

--ping_expose_webserver_url

Controls whether PingImpalaService, PingImpalaHS2Service RPC calls should expose the debug web url to the client or not.

By default this flag is set to true so that the users will not see an empty string when the debug web UI is not available. If the flag is set to false, the RPC calls will return an empty string instead of the real url signalling that the debug web UI is not available.



Note: If the webserver is disabled (--enable_webserver flag is set to false) the RPC calls will return an empty string for the url.

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.
--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.
-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.

Command-Line Option	Configuration File Setting	Explanation
-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.
-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 DEFAULT is used.
--ssl	ssl=true	Enables TLS/SSL for `impala-shell`.
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.
--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.

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 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.

Related Information

"Cloudera Enterprise Connector Documentation"

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 DataHub

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 DataHub Data Mart

Lists an example code to connect to DataHub 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()
```

Related Information

Running Queries on Impala Tables

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 Hue 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 Hue 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 ClustersImpala.
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, Hue, 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 Hue 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.

Related Information

Impala query options

Shut Down Impala

Explains how to gracefully shut down Impala Daemons by first allowing running queries a specified amount of time to complete the process.

The flow to gracefully shut down an Impala Daemon is as follows:

1. The shutdown is initiated.
2. The grace period starts. The Impala Daemon informs other coordinators not to schedule any new queries on it. This allows queries already scheduled to run on this daemon by other coordinators to start executing.
3. The grace period expires.
4. The Impala Daemon continuously checks if there are no queries or fragments running.
5. If there are no queries or fragments running, it shuts down.
6. Otherwise, when it reaches the IMPALA_GRACEFUL_SHUTDOWN_DEADLINE duration, Impala Daemon shuts down.

Once Cloudera Manager initiates the stop/shutdown command, the Impala Daemon starts up the graceful shutdown process, and the process cannot be reverted. However, if you need to change the hard deadline in Cloudera Manager, you can cancel the shutdown command, change the Impala Daemon Graceful Shutdown, and start the shutdown command again.

Procedure

1. Optionally, set the grace period.
   a) In Cloudera Manager, navigate to Impala ServiceConfigurationScopeImpala Daemon.
   b) In the Impala Daemon Command Line Argument Advanced Configuration Snippet (Safety Valve) field, specify the grace period:

      --shutdown_grace_period_s=<new grace period in seconds>

      The default grace period is 2 minutes.

      It is strongly recommended that you use the default value and not change the setting.
2. Optionally, set the hard deadline after which Impala is shut down regardless of whether queries are still running on it.
   a) In Cloudera Manager, navigate to Impala ServiceConfigurationScopeImpala Daemon.
   b) In the Impala Graceful Shutdown Deadline field, specify the time to wait for running queries.

      The default is 60 minutes.

      If you specify 0, Impala will shutdown immediately without waiting for running queries
3. In Cloudera Manager, navigate to Impala ServiceInstances.
4. Click an Impala Daemon role.
5. Click ActionsImpala Daemon Graceful Shutdown.

Related Information

SHUTDOWN statement

Setting Timeouts in Impala

Depending on how busy your cluster is, you might increase or decrease various timeout values. Increase timeouts if Impala is cancelling operations prematurely, when the system is responding slower than usual but the operations are

still successful if given extra time. Decrease timeouts if operations are idle or hanging for long periods, and the idle or hung operations are consuming resources and reducing concurrency.

Setting Timeout and Retries for Thrift Connections to Backend Client

Impala connections to the backend client are subject to failure in cases when the network is momentarily overloaded.

About this task

To avoid failed queries due to transient network problems, you can configure the number of Thrift connection retries using the following option:

Procedure

1. In Cloudera Manager, navigate to Impala serviceConfiguration.
2. In the Impala Daemon Command Line Argument Advanced Configuration Snippet (Safety Valve) field, specify the following.

   To avoid failed queries due to transient network problems, you can configure the number of Thrift connection retries using the following option:

   - The --backend_client_connection_num_retries option specifies the number of times Impala will try connecting to the backend client after the first connection attempt fails. By default, `impalad` will attempt three re-connections before it returns a failure.

   You can configure timeouts for sending and receiving data from the backend client. Therefore, if for some reason a query does not respond, instead of waiting indefinitely for a response, Impala will terminate the connection after a configurable timeout.

   - The --backend_client_rpc_timeout_ms option can be used to specify the number of milliseconds Impala should wait for a response from the backend client before it terminates the connection and signals a failure. The default value for this property is 300000 milliseconds, or 5 minutes.
3. Click Save Changes and restart Impala.

Increasing StateStore Timeout

If you have an extensive Impala schema, for example, with hundreds of databases, tens of thousands of tables, you might encounter timeout errors during startup as the Impala catalog service broadcasts metadata to all the Impala nodes using the StateStore service. To avoid such timeout errors on startup, increase the StateStore timeout value from its default of 10 seconds.

About this task

Increase the timeout value of the StateStore service if you see messages in the impalad log such as:

```
Connection with state-store lost
Trying to re-register with state-store
```

Procedure

1. In Cloudera Manager, navigate to Impala serviceConfiguration.
2. In the search field, type -statestore_subscriber_timeout_seconds.
3. In the StateStoreSubscriber Timeout field, specify a new timeout value larger than the current value.
4. Click Save Changes and restart Impala.

Setting the Idle Query and Idle Session Timeouts

To keep long-running queries or idle sessions from tying up cluster resources, you can set timeout intervals for both individual queries, and entire sessions.

About this task

Procedure

1. In Cloudera Manager, navigate to Impala serviceConfiguration.
2. In the search field, type idle.
3. In the Idle Query Timeout field, specify the time in seconds after which an idle query is cancelled.

   This could be a query whose results were all fetched but was never closed, or one whose results were partially fetched and then the client program stopped requesting further results. This condition is most likely to occur in a client program using the JDBC or ODBC interfaces, rather than in the interactive `impala-shell` interpreter. Once a query is cancelled, the client program cannot retrieve any further results from the query.

   You can reduce the idle query timeout by using the QUERY_TIMEOUT_S query option at the query level. Any non-zero value specified in this field serves as an upper limit for the QUERY_TIMEOUT_S query option.

   The value of 0 disables query timeouts.
4. In the Idle Session Timeout field, specify the time in seconds after which an idle session expires.

   A session is idle when no activity is occurring for any of the queries in that session, and the session has not started any new queries. Once a session is expired, you cannot issue any new query requests to it. The session remains open, but the only operation you can perform is to close it.

   The default value of 0 specifies sessions never expire.

   You can override this setting with the IDLE_SESSION_TIMEOUT query option at the session or query level.
5. Click Save Changes and restart Impala.

Results

Impala checks periodically for idle sessions and queries to cancel. The actual idle time before cancellation might be up to 50% greater than the specified configuration setting. For example, if the timeout setting was 60, the session or query might be cancelled after being idle between 60 and 90 seconds.