Cloudera Runtime 7.1.8

Starting and Stopping Apache Impala

Date published: 2020-11-30
Date modified: 2022-08-25

https://docs.cloudera.com/

Legal Notice

Contents

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. You can control client connections to Impala using the Impala coordinator role available under the Configurations tab for a selected Virtual Warehouse.

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.

You can install the Impala shell on a local computer and use it as a client to connect with an Impala Virtual Warehouse instance. If you are connecting from a node that is already a part of a CDH or CDP cluster, you already have Impala shell and do not need to install it. For more information on installing Impala shell, see the link provided under Related Information.

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.

Command-Line Option	Configuration File Setting	Explanation
--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.
--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.
--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.
-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.

Command-Line Option	Configuration File Setting	Explanation
-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.
--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.
--protocol= *protocol*	N/A	Protocol to use for the connection to Impala. Valid *protocol* values are: • 'hs2-http': Impala-shell uses HTTP transport to speak to the Impala Daemon via the HiveServer2 protocol.

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_CON FIG_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 4.

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

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: ${VAR:*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.

Connecting to a kerberized Impala daemon

Using an impala-shell session you can connect to an impalad daemon to issue queries. When you connect to an impalad, it coordinates the execution of all queries sent to it. You can run impala-shell to connect to a Kerberized Impala instance over HTTP in a cluster.

About this task

Kerberos is an enterprise-grade authentication system Impala supports. Kerberos provides strong security benefits including capabilities that render intercepted authentication packets unusable by an attacker. It virtually eliminates the threat of impersonation by never sending a user's credentials in cleartext over the network. Cloudera recommends using impala-shell with Kerberos authentication for strong security benefits while accessing an Impala instance.

Before you begin

- Locate the hostname that is running the impalad daemon.
- 28000 is the default port impalad daemon uses to transmit commands and receive results from client applications over HTTP using the HiveServer2 protocol. Ensure that this port is open.
- Ensure that the host running impala-shell has a preexisting kinit-cached Kerberos ticket that impala-shell can pass to the impala server automatically without the need for the user to reenter the password.
- To override any client connection errors, you should run the Kinit command to retrieve the Ticket Granting Ticket or to extend it if it has already expired.

Procedure

1. To enable Kerberos in the Impala shell, start the impala-shell command using the -k flag.
2. For impala-shell to communicate with the Impala daemon over HTTP through the HiveServer2 protocol, specify --protocol=hs2-http as the protocol.

```
impala-shell -i xxxx-cdh-7-2-3.vpc.cloudera.com -k --protocol=hs2-http
Starting Impala Shell with Kerberos authentication using Python 2.7.5
Using service name 'impala'
Warning: --connect_timeout_ms is currently ignored with HTTP transport.
Opened TCP connection to xxxx-cdh-7-2-3.vpc.cloudera.com:28000
Connected to xxxx-cdh-7-2-3.vpc.cloudera.com:28000
Server version: impalad version 4.0.0-SNAPSHOT RELEASE (build d18b1c1d3f
7230d330b58928513c20e90bab0153)
```

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 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
```

- 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. 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 to Impala Virtual Warehouse

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

Example

Sample code

```
from impala.dbapi import connect

conn = connect(
    host = "Impala VW endpoint",  # could be coordinator or impala-proxy
    port = 443,
    auth_mechanism = "LDAP",
    use_ssl = True,
    use_http_transport = True,
    http_path = "cliservice",
    user = "ldap_userId",
    password = "ldap_password")
cursor = conn.cursor()
cursor.execute("SELECT * FROM default.emax_temp")
for row in cursor:
    print(row)
cursor.close()
conn.close()
```

Configuring Delegation for Clients

Impala supports user 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.

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 an error.

Procedure

To enable delegation:

1. Log in to the CDP web interface and navigate to the Data Warehouse service.
2. In the Data Warehouse service, click Virtual Warehouses in the left navigation panel.
3. Select the Impala Virtual Warehouse, click options  for the warehouse you want to configure Proxy User.
4. Select Edit.
5. In the Configuration tab, click Impala Coordinator.
6. 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.
7. Click Apply to save the changes.

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 ON by default and can be controlled using the SPOOL_QUERY_RESULTS query option.

- Since the query result spooling is enabled by default, 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 thr
ead-reservation=1
     PLAN-ROOT SINK
     |  mem-estimate=4.00MB mem-reservation=4.00MB spill-buffer=2.00MB thr
ead-reservation=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.

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. Log in to the CDP web interface and navigate to the Data Warehouse service.

2. In the Data Warehouse service, click Virtual Warehouses in the left navigation panel.
3. Select the Impala Virtual Warehouse, click options  for the warehouse you want to set the timeout and retry options.
4. Click Edit and navigate to Impala Coordinatorunder Configurationstab.
5. Using the + sign, specify the following if the options are not already added.

   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.
6. Click Apply 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. Log in to the CDP web interface and navigate to the Data Warehouse service.
2. In the Data Warehouse service, click Virtual Warehouses in the left navigation panel.
3. Select the Impala Virtual Warehouse, click options  for the warehouse you want to set the timeout and retry options.
4. Click Edit and navigate to Impala Coordinatorunder Configurationstab.
5. Using the + sign, specify the following if the option is not already added.

   Specify a new timeout value larger than the current value using the option StateStoreSubscriber Timeout.
6. Click Apply 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. Log in to the CDP web interface and navigate to the Data Warehouse service.
2. In the Data Warehouse service, click Virtual Warehouses in the left navigation panel.
3. Select the Impala Virtual Warehouse, click options  for the warehouse you want to set the timeout and retry options.
4. Click Edit and navigate to Impala Coordinatorunder Configurationstab.
5. In the search field, type idle.
6. 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.
7. 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.
8. Click Apply 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.

Adjusting Heartbeat TCP Timeout Interval

Using the TCP flag, you can prevent the Statestore from waiting indefinitely for a response from the subscribers that fail to respond to the heartbeat RPC within the set period.

About this task

This flag statestore_heartbeat_tcp_timeout_seconds defines the time that may elapse before a heartbeat RPC connection request from a Statestore server to an Impalad or a Catalog server (subscribers) should be considered dead.

You can increase the flag value if you see intermittent heartbeat RPC timeouts listed in the statestore's log. You may find the max value of "statestore.priority-topic-update-durations" metric on the statestore to get an idea of a reasonable value to be used in this flag.

Note: The priority topic updates are only small amounts of data that take little time to process, similar to the heartbeat complexity.

Procedure

1. Log in to the CDP web interface and navigate to the Data Warehouse service.
2. In the Data Warehouse service, click Virtual Warehouses in the left navigation panel.
3. Select the Impala Virtual Warehouse, click options  for the warehouse you want to adjust the Heartbeat TCP Timeout.
4. Click Edit and navigate to Impala Statestoredunder Configurationstab.
5. Using the + sign, specify the following if the options are not already added.
6. In the Add Customs Configuration field, add the flag statestore_heartbeat_tcp_timeout_seconds with an appropriate value.
7. You can also control the maximum number of consecutive heartbeat messages an impalad can miss before being declared failed by the statestore by adding this flag statestore_max_missed_heartbeats. Typically, you will not have to change this value.
8. Click Apply and restart Impala.