Cloudera Runtime 1.5.5

Configuring Apache Impala

Date published: 2020-11-30
Date modified: 2025-06-06

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) through 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. 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
--connect_max_tries	--connect_max_tries=4	Sets the maximum number of attempts to connect to a coordinator. Currently, the maximum number of attempts to connect to a coordinator is hard coded to 4 in hs2-HTTP mode. From this release, you can configure the maximum number of attempts impala-shell can make using this option --connect_max_tries. The default value of this option is 4. After the number of attempts specified through this option, impala-shell returns with error "Not connected to Impala"
--hs2_fp_format	hs2_fp_format=<a Python-based format specification expression which will get parsed and applied to floating-pointcolumn values>	Sets the printing format specification for floating point values when using HS2 protocol. The default behaviour makes the values handled by Python's str() built-in method. Use '16G' to match Beeswax protocol's floating-point output format.
--http_cookie_names	http_cookie_names=KNOX_BACKEND-IMPALA	This parameter has to be set to include the cookie that Knox uses for stateful connections. This is what the Knox enableStickySession=true uses. When using impala-shell and http mode, you must update the client connection string to include the cookie being used for session persistence using the `-http_cookie_names`. This config is needed for Active-Active HA in Impala.
--live_progress	live_progress=true	Prints a progress bar showing roughly the percentage complete for each query. Information is updated interactively as the query progresses.
--disable_live_progress	live_progress=false	Disables live_progress in the interactive mode.
--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.)

Command-Line Option	Configuration File Setting	Explanation
-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.
-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`.
--http_socket_timeout_s	http_socket_timeout_s=*HTTP_SOCKET_TIMEOUT_S*	Sets the timeout in seconds after which the socket will time out if the associated operation cannot be completed. Set to None to disable any timeout. This configurable option is only supported for hs2-http mode and the DEFAULT is NONE.
PATH_TO_CERTIFICATE	ca_cert=*PATH_TO_CERTIFICATE*	The local pathname pointing to the third-party CA certificate, or to a copy of the server certificate for self-signed server certificates. If --ca_cert is not set, `impala-shell` enables TLS/SSL, but does not validate the server certificate. This is useful for connecting to a known-good Impala that is only running over TLS/SSL, when a copy of the certificate is not available (such as when debugging customer installations).
-l	use_ldap=true	Enables LDAP authentication.
-u	user=*USER_NAME*	Supplies the username, when LDAP authentication is enabled by the -l option. (Specify the short username, not the full LDAP distinguished name.) The shell then prompts interactively for the password.
--ldap_password_cmd= *COMMAND*	N/A	Specifies a command to run to retrieve the LDAP password, when LDAP authentication is enabled by the -l option. If the command includes space-separated arguments, enclose the command and its arguments in quotation marks.
-j, --jwt	N/A	Indicates that JWT authentication will be used when this command line option is specified in the impala-shell command.
--jwt_cmd	N/A	Shell command to run to retrieve the JWT to be used for authentication.
--config_file= *PATH_TO_CONFIG_FILE*	N/A	Specifies the path of the file containing `impala-shell` configuration settings. The default is /etc/impalarc. This setting can only be specified on the command line.
--live_progress	N/A	Prints a progress bar showing roughly the percentage complete for each query. The information is updated interactively as the query progresses.
--live_summary	N/A	Prints a detailed report, similar to the SUMMARY command, showing progress details for each phase of query execution. The information is updated interactively as the query progresses.
--var= *VARIABLE_NAME=* *VALUE*	N/A	Defines a substitution variable that can be used within the `impala-shell` session. The variable can be substituted into statements processed by the -q or -f options, or in an interactive shell session. Within a SQL statement, you substitute the value by using the notation ${var:*VARIABLE_NAME*}.

Command-Line Option	Configuration File Setting	Explanation
--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_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 4.

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
```

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.

Command	Explanation
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.

  You can get the Impala daemon and port information from the ⋮ Copy Impala shell Download Command in the UI.
- 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)
```

Setting up ODBC connection from a BI tool

Describes how to connect to Impala Virtual Warehouses using ODBC with your BI tool, with Tableau as an example.

Before you begin

Before you can use Tableau with Impala Virtual Warehouses, you must have created a Database Catalog that is populated with data. You have the option to populate your Database Catalog with sample data when you create it. You must also create an Impala Virtual Warehouse, which is configured to connect to the Database Catalog that is populated with data.

Procedure

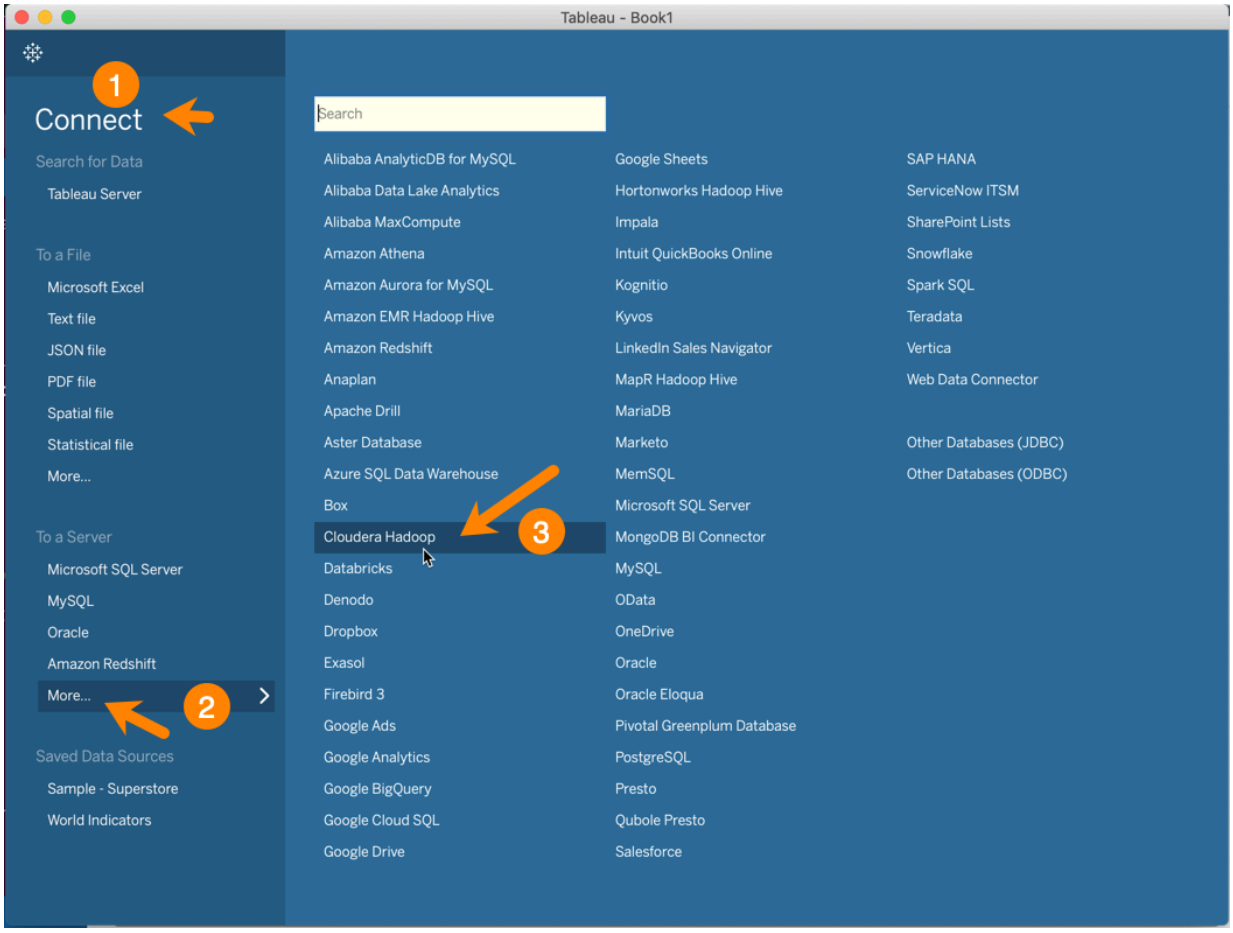
1. Download the latest version of the Impala ODBC driver from Cloudera Downloads page or alternatively, on the **Virtual Warehouses** page, click the options menu for the warehouse you want to connect to your BI tool, and select Download JDBC/ODBC Driver and install it.
2. Install the driver on the local host where you intend to use Tableau Desktop.
3. Log in to the Cloudera web interface and navigate to the **Data Warehouse** service.
4. Click Virtual Warehouse in the left navigation panel.
5. On the **Virtual Warehouses** page, click ⋮ for the Impala warehouse you want to connect to with Tableau, and select Copy JDBC URL:

   This copies the JDBC URL to your system clipboard.
6. Paste the copied JDBC URL into a text file. It should look similar to the following:

   ```
   jdbc:hive2://<your-virtual-warehouse>.<your-environment>.<dwx.company.co
   m>/default;transportMode=http;httpPath=cliservice;ssl=true;retries=3
   ```
7. From the text file where you just pasted the URL, copy the host name from the JDBC URL to your system clipboard. For example, in the URL shown in Step 6, the host name is:

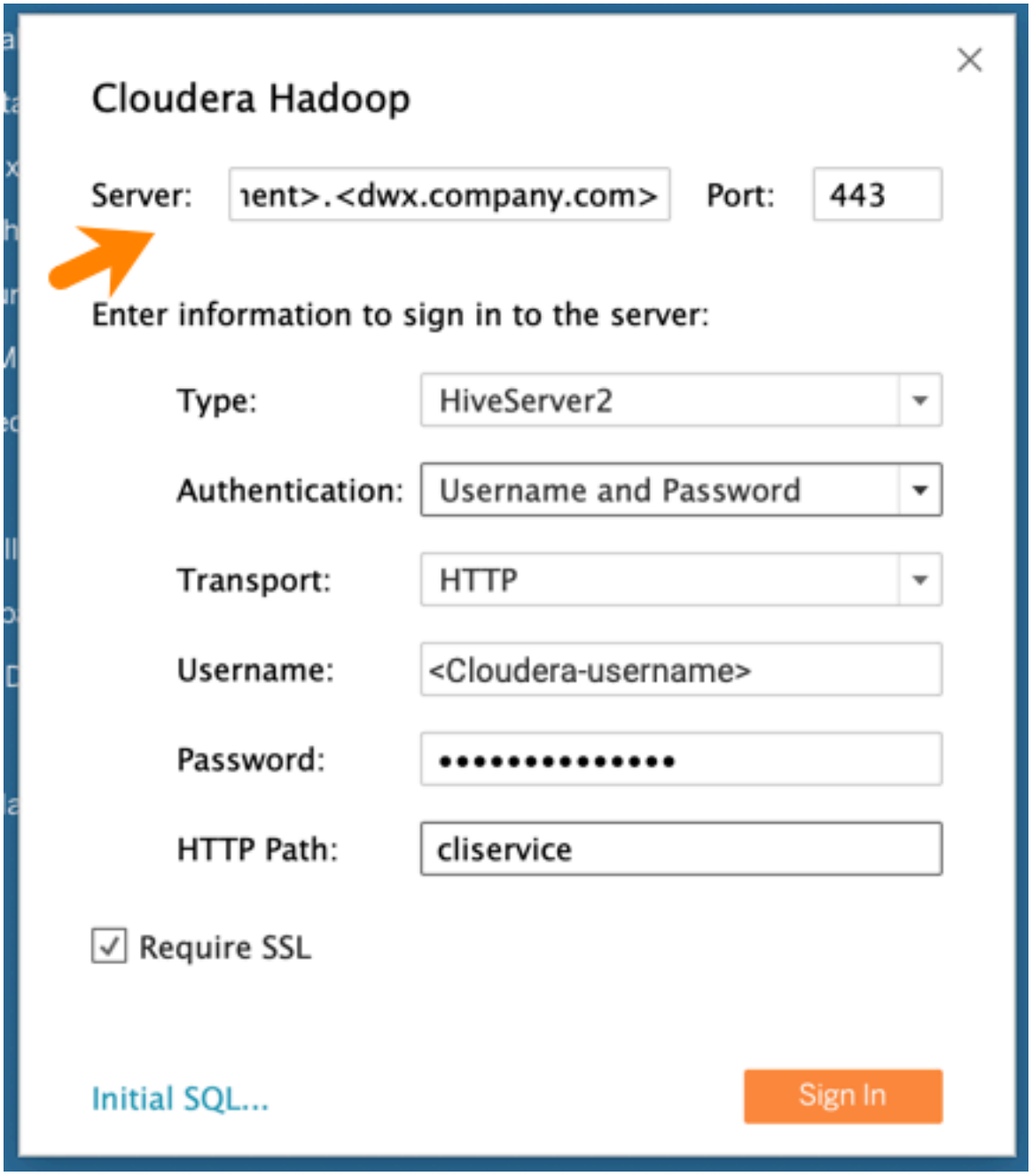
   ```
   <your-virtual-warehouse>.<your-environment>.<dwx.company.com>
   ```

8. Start Tableau and navigate to ConnectMore…Cloudera Hadoop :



This launches the Cloudera Hadoop dialog box.

9. In the Tableau Cloudera Hadoop dialog box, paste the host name you copied to your clipboard in Step 7 into the Server field:



10. Then in the Tableau Cloudera Hadoop dialog box, set the following options:

- Port: 443
- Type: HiveServer2
- Authentication: Username and Password
- Transport: HTTP
- Username: Username to connect to the CDP Data Warehouse service.
- Password: Password to connect to the CDP Data Warehouse service.
- HTTP Path: cliservice
- Require SSL: Make sure this is selected.

11. Click Sign In.

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.

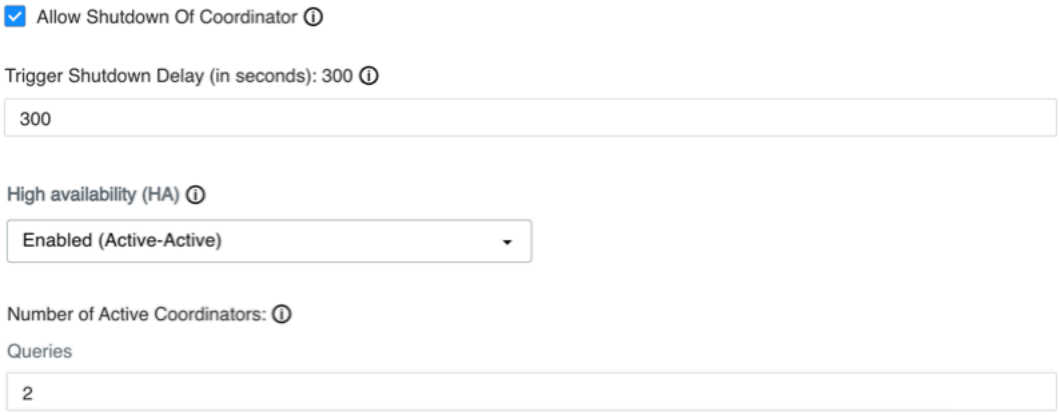
- When result spooling is disabled, a single batch or rows is sent to the PlanRootSink, and then the client must consume that batch before another one can be sent.
- When result spooling is enabled, multiple batches of rows can be sent to the PlanRootSink, and multiple batches can be consumed by the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

Configuring admission control

Learn how to configure admission control in Impala Virtual Warehouses, including enabling and disabling the global admission controller.

Enabling admission control

The admission controller is designed to handle query admission in Impala Virtual Warehouses that have more than one active coordinator. Admission control is enabled by default when a new Impala Virtual Warehouse is created in High Availability (HA) Active-Active



mode.

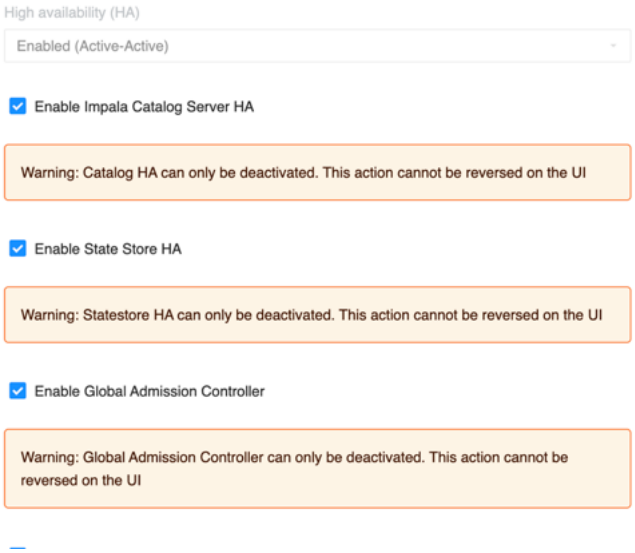
Disabling admission control

Admission control is enabled by default if you have configured an Active-Active Impala coordinator to run in an Impala Virtual Warehouse. However, for debugging or other purposes, you can disable the global admission controller by performing the following steps:

Note: The required role for this action is DWAdmin.

1. Log in to the Cloudera web interface and navigate to the Cloudera Data Warehouse service. The **Overview** page displays.
2. In the **Overview** page under **Virtual Warehouses** tab, select the Virtual Warehouse where you want to disable admission controller, click the icon and select the Edit action form the list of actions.
3. In the Sizing And Scaling tab, view the properties you can adjust to disable the service.

4. Deselect the Enable Global Admission Controller checkbox.



Attention: Global Admission Controller can only be deactivated. This action cannot be reversed on the UI.

5. Click the **Apply changes** button at the bottom of the page to apply your changes.

Configuring Impala

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

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

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

Short-Circuit Reads

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

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

To Enable Short-Circuit Reads

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

1. In Cloudera Manager Clusters select the Impala service, for example, IMPALA.
2. On the Configuration tab, search for dfs.client.read.shortcircuit.
3. Accept the default (enabled), or check to enable the dfs.client.read.shortcircuit property to read the HDFS file blocks directly.

4. Do the above for HDFS service too by clicking Cloudera Manager Clusters and by selecting the HDFS service.
5. Save the changes and Restart the service.

To configure DataNodes for short-circuit reads:

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

Block Location Tracking

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

To enable block location tracking:

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

Native Checksumming

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

To enable native checksumming:

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

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

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