HiveServer2 Clients

- Beeline – Command Line Shell
  - Beeline Example
  - Beeline Commands
  - Beeline Properties
  - Beeline Hive Commands
  - Beeline Command Options
  - Output Formats
    - table
    - vertical
    - xmlattr
    - xmlelements
    - json
    - jsonfile
    - Separated-Value Output Formats
      - csv2, tsv2, dsv
        - Quoting in csv2, tsv2 and dsv Formats
      - csv, tsv
  - HiveServer2 Logging
  - Cancelling the Query
  - Background Query in Terminal Script
- JDBC
  - Connection URLs
    - Connection URL Format
    - Connection URL for Remote or Embedded Mode
    - Connection URL When HiveServer2 Is Running in HTTP Mode
    - Connection URL When SSL Is Enabled in HiveServer2
    - Connection URL When ZooKeeper Service Discovery Is Enabled
    - Named Connection URLs
    - Reconnecting
    - Using hive-site.xml to automatically connect to HiveServer2
    - Using beeline-site.xml to automatically connect to HiveServer2
    - Using JDBC
      - JDBC Client Sample Code
      - Running the JDBC Sample Code
  - JDBC Data Types
  - JDBC Client Setup for a Secure Cluster
    - Multi-User Scenarios and Programmatic Login to Kerberos KDC
      - Using Kerberos with a Pre-Authenticated Subject
  - JDBC Fetch Size
- Python Client
- Ruby Client
- Integration with SQuirrel SQL Client
- Integration with SQL Developer
- Integration with DbVisSoftware's DbVisualizer
- Advanced Features for Integration with Other Tools
  - Supporting Cookie Replay in HTTP Mode
  - Using 2-way SSL in HTTP Mode
  - Passing HTTP Header Key/Value Pairs via JDBC Driver
  - Passing Custom HTTP Cookie Key/Value Pairs via JDBC Driver

This page describes the different clients supported by HiveServer2. Other documentation for HiveServer2 includes:

- HiveServer2 Overview
- Setting Up HiveServer2
- Hive Configuration Properties: HiveServer2

> **Version**
>
> Introduced in Hive version 0.11. See HIVE-2935.

Beeline – Command Line Shell

HiveServer2 supports a command shell Beeline that works with HiveServer2. It's a JDBC client that is based on the SQLLine CLI (http://sqlline.sourceforge.net/). There's detailed documentation of SQLLine which is applicable to Beeline as well.

Replacing the Implementation of Hive CLI Using Beeline

The Beeline shell works in both embedded mode as well as remote mode. In the embedded mode, it runs an embedded Hive (similar to Hive CLI) whereas remote mode is for connecting to a separate HiveServer2 process over Thrift. Starting in Hive 0.14, when Beeline is used with HiveServer2, it also prints the log messages from HiveServer2 for queries it executes to STDERR. Remote HiveServer2 mode is recommended for production use, as it is more secure and doesn't require direct HDFS/metastore access to be granted for users.

In remote mode HiveServer2 only accepts valid Thrift calls – even in HTTP mode, the message body contains Thrift payloads.

Beeline Example

```
% bin/beeline
Hive version 0.11.0-SNAPSHOT by Apache
beeline> !connect jdbc:hive2://localhost:10000 scott tiger
!connect jdbc:hive2://localhost:10000 scott tiger
Connecting to jdbc:hive2://localhost:10000
Connected to: Hive (version 0.10.0)
Driver: Hive (version 0.10.0-SNAPSHOT)
Transaction isolation: TRANSACTION_REPEATABLE_READ
0: jdbc:hive2://localhost:10000> show tables;
show tables;
+-------------------+
|     tab_name      |
+-------------------+
| primitives        |
| src               |
| src1              |
| src_json          |
| src_sequencefile  |
| src_thrift        |
| srcbucket         |
| srcbucket2        |
| srcpart           |
+-------------------+
9 rows selected (1.079 seconds)
```

You can also specify the connection parameters on command line. This means you can find the command with the connection string from your UNIX shell history.

```
% beeline -u jdbc:hive2://localhost:10000/default -n scott -w password_file
Hive version 0.11.0-SNAPSHOT by Apache

Connecting to jdbc:hive2://localhost:10000/default
```

Beeline with NoSASL connection

If you'd like to connect via NOSASL mode, you must specify the authentication mode explicitly:

```
% bin/beeline
beeline> !connect jdbc:hive2://<host>:<port>/<db>;auth=noSasl hiveuser pass
```

Beeline Commands

Command	Description
!<SQLLine command>	List of SQLLine commands available at http://sqlline.sourceforge.net/. Example: `!quit` exits the Beeline client.
!delimiter	Set the delimiter for queries written in Beeline. Multi-character delimiters are allowed, but quotation marks, slashes, and -- are not allowed. Defaults to ; Usage: `!delimiter $$` Version: 3.0.0 (HIVE-10865)

Beeline Properties

Property	Description
fetchsize	Standard JDBC enables you to specify the number of rows fetched with each database round-trip for a query, and this number is referred to as the fetch size. Setting the fetch size in Beeline overrides the JDBC driver's default fetch size and affects subsequent statements executed in the current session. 1. A value of -1 instructs Beeline to use the JDBC driver's default fetch size (default) 2. A value of zero or more is passed to the JDBC driver for each statement 3. Any other negative value will throw an Exception Usage: `!set fetchsize 200` Version: 4.0.0 (HIVE-22853)

Beeline Hive Commands

Hive specific commands (same as Hive CLI commands) can be run from Beeline, when the Hive JDBC driver is used.

Use "`;`" (semicolon) to terminate commands. Comments in scripts can be specified using the "`--`" prefix.

Command	Description
reset	Resets the configuration to the default values.
reset <key>	Resets the value of a particular configuration variable (key) to the default value. **Note:** If you misspell the variable name, Beeline will not show an error.
set <key>=<value>	Sets the value of a particular configuration variable (key). **Note:** If you misspell the variable name, Beeline will not show an error.
set	Prints a list of configuration variables that are overridden by the user or Hive.
set -v	Prints all Hadoop and Hive configuration variables.
add FILE[S] <filepath> <filepath>* add JAR[S] <filepath> <filepath>* add ARCHIVE[S] <filepath> <filepath>*	Adds one or more files, jars, or archives to the list of resources in the distributed cache. See Hive Resources for more information.
add FILE[S] <ivyurl> <ivyurl>* add JAR [S] <ivyurl> <ivyurl>* add ARCHIVE[S] <ivyurl> <ivyurl>*	As of Hive 1.2.0, adds one or more files, jars or archives to the list of resources in the distributed cache using an Ivy URL of the form ivy://group:module:version?query_string. See Hive Resources for more information.
list FILE[S] list JAR[S] list ARCHIVE[S]	Lists the resources already added to the distributed cache. See Hive Resources for more information. (As of Hive 0.14.0: HIVE-7592).
list FILE[S] <filepath>* list JAR[S] <filepath>* list ARCHIVE[S] <filepath>*	Checks whether the given resources are already added to the distributed cache or not. See Hive Resources for more information.
delete FILE[S] <filepath>* delete JAR[S] <filepath>* delete ARCHIVE [S] <filepath>*	Removes the resource(s) from the distributed cache.

delete FILE[S] <ivyurl> <ivyurl>* delete JAR[S] <ivyurl> <ivyurl>* delete ARCHIVE [S] <ivyurl> <ivyurl>*	As of Hive 1.2.0, removes the resource(s) which were added using the <ivyurl> from the distributed cache. See Hive Resources for more information.
reload	As of Hive 0.14.0, makes HiveServer2 aware of any jar changes in the path specified by the configuration parameter hive.reloadable.aux.jars.path (without needing to restart HiveServer2). The changes can be adding, removing, or updating jar files.
dfs <dfs command>	Executes a dfs command.
<query string>	Executes a Hive query and prints results to standard output.

Beeline Command Options

The Beeline CLI supports these command line options:

Option	Description
-u <*database URL*>	The JDBC URL to connect to. Special characters in parameter values should be encoded with URL encoding if needed. Usage: `beeline -u` *db_URL*
-r	Reconnect to last used URL (if a user has previously used `!connect` to a URL and used `!save` to a beeline.properties file). Usage: `beeline -r` Version: 2.1.0 (HIVE-13670)
-n <*username*>	The username to connect as. Usage: `beeline -n` *valid_user*
-p <*password*>	The password to connect as. Usage: `beeline -p` *valid_password* Optional password mode: Starting Hive 2.2.0 (HIVE-13589) the argument for -p option is optional. Usage : beeline -p [valid_password] If the password is not provided after -p Beeline will prompt for the password while initiating the connection. When password is provided Beeline uses it initiate the connection without prompting.
-d <*driver class*>	The driver class to use. Usage: `beeline -d` *driver_class*
-e <*query*>	Query that should be executed. Double or single quotes enclose the query string. This option can be specified multiple times. Usage: `beeline -e "`*query_string*" Support to run multiple SQL statements separated by semicolons in a single *query_string*: 1.2.0 (HIVE-9877) Bug fix (null pointer exception): 0.13.0 (HIVE-5765) Bug fix (--headerInterval not honored): 0.14.0 (HIVE-7647) Bug fix (running `-e` in background): 1.3.0 and 2.0.0 (HIVE-6758); workaround available for earlier versions
-f <*file*>	Script file that should be executed. Usage: `beeline -f` *filepath* Version: 0.12.0 (HIVE-4268) Note: If the script contains tabs, query compilation fails in version 0.12.0. This bug is fixed in version 0.13.0 (HIVE-6359). Bug fix (running `-f` in background): 1.3.0 and 2.0.0 (HIVE-6758); workaround available for earlier versions

-i (or) **--init** <file or files>	The init files for initialization Usage: `beeline -i /tmp/initfile` Single file: Version: 0.14.0 (HIVE-6561) Multiple files: Version: 2.1.0 (HIVE-11336)
-w (or) **--password-file** <password file>	The password file to read password from. Version: 1.2.0 (HIVE-7175)
-a (or) **--authType** <auth type>	The authentication type passed to the jdbc as an auth property Version: 0.13.0 (HIVE-5155)
--property-file <file>	File to read configuration properties from Usage: `beeline --property-file /tmp/a` Version: 2.2.0 (HIVE-13964)
--hiveconf *property=value*	Use *value* for the given configuration property. Properties that are listed in hive.conf.restricted.list cannot be reset with **hiveconf** (see Restricted List and Whitelist). Usage: `beeline --hiveconf prop1=value1` Version: 0.13.0 (HIVE-6173)
--hivevar *name=value*	Hive variable name and value. This is a Hive-specific setting in which variables can be set at the session level and referenced in Hive commands or queries. Usage: `beeline --hivevar var1=value1`
--color=[true/false]	Control whether color is used for display. Default is false. Usage: `beeline --color=true` (Not supported for Separated-Value Output formats. See HIVE-9770)
--showHeader=[true/false]	Show column names in query results (true) or not (false). Default is true. Usage: `beeline --showHeader=false`
--headerInterval=ROWS	The interval for redisplaying column headers, in number of rows, when **outputformat** is table. Default is 100. Usage: `beeline --headerInterval=50` (Not supported for Separated-Value Output formats. See HIVE-9770)
--fastConnect=[true/false]	When connecting, skip building a list of all tables and columns for tab-completion of HiveQL statements (true) or build the list (false). Default is true. Usage: `beeline --fastConnect=false`
--autoCommit=[true/false]	Enable/disable automatic transaction commit. Default is false. Usage: `beeline --autoCommit=true`
--verbose=[true/false]	Show verbose error messages and debug information (true) or do not show (false). Default is false. Usage: `beeline --verbose=true`
--showWarnings=[true/false]	Display warnings that are reported on the connection after issuing any HiveQL commands. Default is false. Usage: `beeline --showWarnings=true`

--showDbInPrompt=[true/false]	Display the current database name in prompt. Default is false. Usage: `beeline --showDbInPrompt=true` Version: 2.2.0 (HIVE-14123)
--showNestedErrs=[true/false]	Display nested errors. Default is false. Usage: `beeline --showNestedErrs=true`
--numberFormat=[pattern]	Format numbers using a DecimalFormat pattern. Usage: `beeline --numberFormat="#,###,##0.00"`
--force=[true/false]	Continue running script even after errors (true) or do not continue (false). Default is false. Usage: `beeline--force=true`
--maxWidth= MAXWIDTH	The maximum width to display before truncating data, in characters, when **outputformat** is table. Default is to query the terminal for current width, then fall back to 80. Usage: `beeline --maxWidth=150`
--maxColumnWidth=MAXCOLWIDTH	The maximum column width, in characters, when **outputformat** is table. Default is 50 in Hive version 2.2.0+ (see HIVE-14135) or 15 in earlier versions. Usage: `beeline --maxColumnWidth=25`
--silent=[true/false]	Reduce the amount of informational messages displayed (true) or not (false). It also stops displaying the log messages for the query from HiveServer2 (Hive 0.14 and later) and the HiveQL commands (Hive 1.2.0 and later). Default is false. Usage: `beeline --silent=true`
--autosave=[true/false]	Automatically save preferences (true) or do not autosave (false). Default is false. Usage: `beeline --autosave=true`
--outputformat=[table/vertical/csv/tsv/dsv/csv2/tsv2]	Format mode for result display. Default is table. See Separated-Value Output Formats below for description of recommended sv options. Usage: `beeline --outputformat=tsv` Version: dsv/csv2/tsv2 added in 0.14.0 (HIVE-8615)
--truncateTable=[true/false]	If true, truncates table column in the console when it exceeds console length. Version: 0.14.0 (HIVE-6928)
--delimiterForDSV= DELIMITER	The delimiter for delimiter-separated values output format. Default is '|' character. Version: 0.14.0 (HIVE-7390)
--isolation= LEVEL	Set the transaction isolation level to TRANSACTION_READ_COMMITTED or TRANSACTION_SERIALIZABLE. See the "Field Detail" section in the Java Connection documentation. Usage: `beeline --isolation=TRANSACTION_SERIALIZABLE`
--nullemptystring=[true/false]	Use historic behavior of printing null as empty string (true) or use current behavior of printing null as NULL (false). Default is false. Usage: `beeline --nullemptystring=false` Version: 0.13.0 (HIVE-4485)
--incremental=[true/false]	Defaults to `true` from Hive 2.3 onwards, before it defaulted to `false.` When set to `false`, the entire result set is fetched and buffered before being displayed, yielding optimal display column sizing. When set to `true`, result rows are displayed immediately as they are fetched, yielding lower latency and memory usage at the price of extra display column padding. Setting `--incremental=true` is recommended if you encounter an OutOfMemory on the client side (due to the fetched result set size being large).
--incrementalBufferRows=NUMROWS	The number of rows to buffer when printing rows on stdout, defaults to 1000; only applicable if `--incremental=true` and `--outputformat=table` Usage: `beeline --incrementalBufferRows=1000` Version: 2.3.0 (HIVE-14170)

--maxHistoryRows=NUMROWS	The maximum number of rows to store Beeline history. Version: 2.3.0 (HIVE-15166)
--delimiter=;	Set the delimiter for queries written in Beeline. Multi-char delimiters are allowed, but quotation marks, slashes, and -- are not allowed. Defaults to ; Usage: `beeline --delimiter=$$` Version: 3.0.0 (HIVE-10865)
--convertBinaryArrayToString=[true/false]	Display binary column data as a string using the platform's default character set. The default behavior (false) is to display binary data using: `Arrays.toString(byte[] columnValue)` Version: 3.0.0 (HIVE-14786) Display binary column data as a string using the UTF-8 character set. The default behavior (false) is to display binary data using Base64 encoding without padding. Version: 4.0.0 (HIVE-23856) Usage: `beeline --convertBinaryArrayToString=true`
--help	Display a usage message. Usage: `beeline --help`

Output Formats

In Beeline, the result can be displayed in different formats. The format mode can be set with the `outputformat` option.

The following output formats are supported:

- table
- vertical
- xmlattr
- xmlelements
- HiveServer2 Clients#json
- HiveServer2 Clients#jsonfile
- separated-value formats (csv, tsv, csv2, tsv2, dsv)

table

The result is displayed in a table. A row of the result corresponds to a row in the table and the values in one row are displayed in separate columns in the table.
This is the default format mode.

Result of the query *`select id, value, comment from test_table`*

```
+-----+---------+-----------------+
| id  |  value  |     comment     |
+-----+---------+-----------------+
| 1   | Value1  | Test comment 1  |
| 2   | Value2  | Test comment 2  |
| 3   | Value3  | Test comment 3  |
+-----+---------+-----------------+
```

vertical

Each row of the result is displayed in a block of key-value format, where the keys are the names of the columns.

Result of the query *`select id, value, comment from test_table`*

```
id       1
value    Value1
comment  Test comment 1

id       2
value    Value2
comment  Test comment 2

id       3
value    Value3
comment  Test comment 3
```

xmlattr

The result is displayed in an XML format where each row is a "result" element in the XML.
The values of a row are displayed as attributes on the "result" element. The names of the attributes are the names of the columns.

Result of the query *`select id, value, comment from test_table`*

```
<resultset>
  <result id="1" value="Value1" comment="Test comment 1"/>
  <result id="2" value="Value2" comment="Test comment 2"/>
  <result id="3" value="Value3" comment="Test comment 3"/>
</resultset>
```

xmlelements

The result is displayed in an XML format where each row is a "result" element in the XML. The values of a row are displayed as child elements of the result element.

Result of the query *`select id, value, comment from test_table`*

```
<resultset>
  <result>
    <id>1</id>
    <value>Value1</value>
    <comment>Test comment 1</comment>
  </result>
  <result>
    <id>2</id>
    <value>Value2</value>
    <comment>Test comment 2</comment>
  </result>
  <result>
    <id>3</id>
    <value>Value3</value>
    <comment>Test comment 3</comment>
  </result>
</resultset>
```

json

(Hive 4.0) The result is displayed in JSON format where each row is a "result" element in the JSON array "resultset".

Result of the query *`select `String`, `Int`, `Decimal`, `Bool`, `Null`, `Binary` from test_table`*

```
{"resultset":[{"String":"aaa","Int":1,"Decimal":3.14,"Bool":true,"Null":null,"Binary":"SGVsbG8sIFdvcmxkIQ"},
{"String":"bbb","Int":2,"Decimal":2.718,"Bool":false,"Null":null,"Binary":"RWFzdGVyCgllZ2cu"}]}
```

jsonfile

(Hive 4.0) The result is displayed in JSON format where each row is a distinct JSON object. This matches the expected format for a table created as JSONFILE format.

Result of the query *`select `String`, `Int`, `Decimal`, `Bool`, `Null`, `Binary` from test_table`*

```
{"String":"aaa","Int":1,"Decimal":3.14,"Bool":true,"Null":null,"Binary":"SGVsbG8sIFdvcmxkIQ"}
{"String":"bbb","Int":2,"Decimal":2.718,"Bool":false,"Null":null,"Binary":"RWFzdGVyCgllZ2cu"}
```

Separated-Value Output Formats

The values of a row are separated by different delimiters.
There are five separated-value output formats available: csv, tsv, csv2, tsv2 and dsv.

csv2, tsv2, dsv

Starting with Hive 0.14 there are improved SV output formats available, namely dsv, csv2 and tsv2.
These three formats differ only with the delimiter between cells, which is comma for csv2, tab for tsv2, and configurable for dsv.

For the dsv format, the delimiter can be set with the `delimiterForDSV` option. The default delimiter is '|'.
Please be aware that only single character delimiters are supported.

Result of the query *`select id, value, comment from test_table`*

csv2

```
id,value,comment
1,Value1,Test comment 1
2,Value2,Test comment 2
3,Value3,Test comment 3
```

tsv2

```
id        value        comment
1        Value1        Test comment 1
2        Value2        Test comment 2
3        Value3        Test comment 3
```

dsv (the delimiter is |)

```
id|value|comment
1|Value1|Test comment 1
2|Value2|Test comment 2
3|Value3|Test comment 3
```

Quoting in csv2, tsv2 and dsv Formats

If quoting is not disabled, double quotes are added around a value if it contains special characters (such as the delimiter or double quote character) or spans multiple lines.
Embedded double quotes are escaped with a preceding double quote.

The quoting can be disabled by setting the `disable.quoting.for.sv` system variable to true.
If the quoting is disabled, no double quotes are added around the values (even if they contains special characters) and the embedded double quotes are not escaped.
By default, the quoting is disabled.

Result of the query *`select id, value, comment from test_table`*

csv2, quoting is enabled

```
id,value,comment
1,"Value,1",Value contains comma
2,"Value""2",Value contains double quote
3,Value'3,Value contains single quote
```

csv2, quoting is disabled

```
id,value,comment
1,Value,1,Value contains comma
2,Value"2,Value contains double quote
3,Value'3,Value contains single quote
```

csv, tsv

These two formats differ only with the delimiter between values, which is comma for csv and tab for tsv.
The values are always surrounded with single quote characters, even if the quoting is disabled by the `disable.quoting.for.sv` system variable.
These output formats don't escape the embedded single quotes.
Please be aware that these output formats are deprecated and only maintained for backward compatibility.

Result of the query *`select id, value, comment from test_table`*

csv

```
'id','value','comment'
'1','Value1','Test comment 1'
'2','Value2','Test comment 2'
'3','Value3','Test comment 3'
```

tsv

```
'id'        'value'        'comment'
'1'        'Value1'        'Test comment 1'
'2'        'Value2'        'Test comment 2'
'3'        'Value3'        'Test comment 3'
```

HiveServer2 Logging

Starting with Hive 0.14.0, HiveServer2 operation logs are available for Beeline clients. These parameters configure logging:

- hive.server2.logging.operation.enabled
- hive.server2.logging.operation.log.location
- hive.server2.logging.operation.verbose (Hive 0.14 to 1.1)
- hive.server2.logging.operation.level (Hive 1.2 onward)

HIVE-11488 (Hive 2.0.0) adds the support of logging queryId and sessionId to HiveServer2 log file. To enable that, edit/add %X{queryId} and %X{sessionId} to the pattern format string of the logging configuration file.

Cancelling the Query

When a user enters `CTRL+C` on the Beeline shell, if there is a query which is running at the same time then Beeline attempts to cancel the query while closing the socket connection to HiveServer2. This behavior is enabled only when `hive.server2.close.session.on.disconnect` is set to `true`. Starting from Hive 2.2.0 (HIVE-15626) Beeline does not exit the command line shell when the running query is being cancelled as a user enters `CTRL+C`. If the user wishes to exit the shell they can enter `CTRL+C` for the second time while the query is being cancelled. However, if there is no query currently running, the first `CTRL+C` will exit the Beeline shell. This behavior is similar to how the Hive CLI handles `CTRL+C`.

`!quit` is the recommended command to exit the Beeline shell.

Background Query in Terminal Script

Beeline can be run disconnected from a terminal for batch processing and automation scripts using commands such as *nohup* and *disown*.

Some versions of Beeline client may require a workaround to allow the *nohup* command to correctly put the Beeline process in the background without stopping it. See HIVE-11717, HIVE-6758.

The following environment variable can be updated:

```
export HADOOP_CLIENT_OPTS="$HADOOP_CLIENT_OPTS -Djline.terminal=jline.UnsupportedTerminal"
```

Running with nohangup (nohup) and ampersand (&) will place the process in the background and allow the terminal to disconnect while keeping the Beeline process running.

```
nohup beeline --silent=true --showHeader=true --outputformat=dsv -f query.hql </dev/null > /tmp/output.log 2>
/tmp/error.log &
```

JDBC

HiveServer2 has a JDBC driver. It supports both embedded and remote access to HiveServer2. Remote HiveServer2 mode is recommended for production use, as it is more secure and doesn't require direct HDFS/metastore access to be granted for users.

Connection URLs

Connection URL Format

The HiveServer2 URL is a string with the following syntax:

`jdbc:hive2://<host1>:<port1>,<host2>:<port2>/dbName;initFile=<file>;sess_var_list?hive_conf_list#hive_var_list`

where

- *<host1>:<port1>,<host2>:<port2>* is a server instance or a comma separated list of server instances to connect to (if dynamic service discovery is enabled). If empty, the embedded server will be used.
- *dbName* is the name of the initial database.
- *<file>* is the path of init script file (Hive 2.2.0 and later). This script file is written with SQL statements which will be executed automatically after connection. This option can be empty.
- *sess_var_list* is a semicolon separated list of key=value pairs of session variables (e.g., `user=foo;password=bar`).
- *hive_conf_list* is a semicolon separated list of key=value pairs of Hive configuration variables for this session
- *hive_var_list* is a semicolon separated list of key=value pairs of Hive variables for this session.

Special characters in *sess_var_list, hive_conf_list, hive_var_list* parameter values should be encoded with URL encoding if needed.

Connection URL for Remote or Embedded Mode

The JDBC connection URL format has the prefix `jdbc:hive2://` and the Driver class is `org.apache.hive.jdbc.HiveDriver`. Note that this is different from the old HiveServer.

- For a remote server, the URL format is `jdbc:hive2://<host>:<port>/<db>;initFile=<file>` (default port for HiveServer2 is 10000).
- For an embedded server, the URL format is `jdbc:hive2:///;initFile=<file>` (no host or port).

The `initFile` option is available in Hive 2.2.0 and later releases.

Connection URL When HiveServer2 Is Running in HTTP Mode

JDBC connection URL: `jdbc:hive2://<host>:<port>/<db>;transportMode=http;httpPath=<http_endpoint>`, where:

- `<http_endpoint>` is the corresponding HTTP endpoint configured in hive-site.xml. Default value is `cliservice`.
- Default port for HTTP transport mode is 10001.

> **Versions earlier than 0.14**
>
> In versions earlier than 0.14 these parameters used to be called `hive.server2.transport.mode` and `hive.server2.thrift.http.path` respectively and were part of the *hive_conf_list*. These versions have been deprecated in favour of the new versions (which are part of the *sess_var_list*) but continue to work for now.

Connection URL When SSL Is Enabled in HiveServer2

JDBC connection URL: `jdbc:hive2://<host>:<port>/<db>;ssl=true;sslTrustStore=<trust_store_path>;trustStorePassword=<trust_store_password>`, where:

- <trust_store_path> is the path where client's truststore file lives.
- <trust_store_password> is the password to access the truststore.

In HTTP mode: `jdbc:hive2://<host>:<port>/<db>;ssl=true;sslTrustStore=<trust_store_path>;trustStorePassword=<trust_store_password>;transportMode=http;httpPath=<http_endpoint>`.

For versions earlier than 0.14, see the version note above.

Connection URL When ZooKeeper Service Discovery Is Enabled

ZooKeeper-based service discovery introduced in Hive 0.14.0 (HIVE-7935) enables high availability and rolling upgrade for HiveServer2. A JDBC URL that specifies <zookeeper quorum> needs to be used to make use of these features.

With further changes in Hive 2.0.0 and 1.3.0 (unreleased, HIVE-11581), none of the additional configuration parameters such as authentication mode, transport mode, or SSL parameters need to be specified, as they are retrieved from the ZooKeeper entries along with the hostname.

The JDBC connection URL: `jdbc:hive2://<zookeeper quorum>/;serviceDiscoveryMode=zooKeeper; zooKeeperNamespace=hiveserver2` .

The <zookeeper quorum> is the same as the value of hive.zookeeper.quorum configuration parameter in hive-site.xml/hivserver2-site.xml used by HiveServer2.

Additional runtime parameters needed for querying can be provided within the URL as follows, by appending it as a ?<option> as before.

The JDBC connection URL: `jdbc:hive2://<zookeeper quorum>/;serviceDiscoveryMode=zooKeeper; zooKeeperNamespace=hiveserver2?tez.queue.name=hive1&hive.server2.thrift.resultset.serialize.in.tasks=true`

Named Connection URLs

As of Hive 2.1.0 (HIVE-13670), Beeline now also supports named URL connect strings via usage of environment variables. If you try to do a `!connect` to a name that does not look like a URL, then Beeline will attempt to see if there is an environment variable called BEELINE_URL_<name>. For instance, if you specify `!connect blue`, it will look for BEELINE_URL_BLUE, and use that to connect. This should make it easier for system administrators to specify environment variables for users, and users need not type in the full URL each time to connect.

Reconnecting

Traditionally, `!reconnect` has worked to refresh a connection that has already been established. It is not able to do a fresh connect after `!close` has been run. As of Hive 2.1.0 (HIVE-13670), Beeline remembers the last URL successfully connected to in a session, and is able to reconnect even after a `!close` has been run. In addition, if a user does a `!save`, then this is saved in the beeline.properties file, which then allows `!reconnect` to connect to this saved last-connected-to URL across multiple Beeline sessions. This also allows the use of `beeline -r` from the command line to do a reconnect on startup.

Using hive-site.xml to automatically connect to HiveServer2

As of Hive 2.2.0 (HIVE-14063), Beeline adds support to use the hive-site.xml present in the classpath to automatically generate a connection URL based on the configuration properties in hive-site.xml and an additional user configuration file. Not all the URL properties can be derived from hive-site.xml and hence in order to use this feature user must create a configuration file called “beeline-hs2-connection.xml” which is a Hadoop XML format file. This file is used to provide user-specific connection properties for the connection URL. Beeline looks for this configuration file in ${user.home}/.beeline/ (Unix based OS) or ${user.home}\beeline\ directory (in case of Windows). If the file is not found in the above locations Beeline looks for it in ${HIVE_CONF_DIR} location and /etc/hive/conf (check HIVE-16335 which fixes this location from /etc/conf/hive in Hive 2.2.0) in that order. Once the file is found, Beeline uses beeline-hs2-connection.xml in conjunction with the hive-site.xml in the class path to determine the connection URL.

The URL connection properties in beeline-hs2-connection.xml must have the prefix “beeline.hs2.connection.” followed by the URL property name. For example in order to provide the property ssl the property key in the beeline-hs2-connection.xml should be “beeline.hs2.connection.ssl”. The sample beeline.hs2.connection.xml below provides the value of user and password for the Beeline connection URL. In this case the rest of the properties like HS2 hostname and port information, Kerberos configuration properties, SSL properties, transport mode, etc., are picked up using the hive-site.xml in the class path. If the password is empty beeline.hs2.connection.password property should be removed. In most cases the below configuration values in beeline-hs2-connection.xml and the correct hive-site.xml in classpath should be sufficient to make the connection to the HiveServer2.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
  <name>beeline.hs2.connection.user</name>
  <value>hive</value>
</property>
<property>
  <name>beeline.hs2.connection.password</name>
  <value>hive</value>
</property>
</configuration>
```

In case of properties which are present in both beeline-hs2-connection.xml and hive-site.xml, the property value derived from beeline-hs2-connection.xml takes precedence. For example in the below beeline-hs2-connection.xml file provides the value of principal for Beeline connection in a Kerberos enabled environment. In this case the property value for beeline.hs2.connection.principal overrides the value of HiveConf.ConfVars. HIVE_SERVER2_KERBEROS_PRINCIPAL from hive-site.xml as far as connection URL is concerned.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
  <name>beeline.hs2.connection.hosts</name>
  <value>localhost:10000</value>
</property>
<property>
  <name>beeline.hs2.connection.principal</name>
  <value>hive/dummy-hostname@domain.com</value>
</property>
</configuration>
```

In case of properties beeline.hs2.connection.hosts, beeline.hs2.connection.hiveconf and beeline.hs2.connection.hivevar the property value is a comma-separated list of values. For example the following beeline-hs2-connection.xml provides the hiveconf and hivevar values in a comma separated format.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
  <name>beeline.hs2.connection.user</name>
  <value>hive</value>
</property>
<property>
  <name>beeline.hs2.connection.hiveconf</name>
  <value>hive.cli.print.current.db=true, hive.cli.print.header=true</value>
</property>
<property>
  <name>beeline.hs2.connection.hivevar</name>
  <value>testVarName1=value1, testVarName2=value2</value>
</property>
</configuration>
```

When the beeline-hs2-connection.xml is present and when no other arguments are provided, Beeline automatically connects to the URL generated using configuration files. When connection arguments (-u, -n or -p) are provided, Beeline uses them and does not use beeline-hs2-connection.xml to automatically connect. Removing or renaming the beeline-hs2-connection.xml disables this feature.

Using beeline-site.xml to automatically connect to HiveServer2

In addition to the above method of using hive-site.xml and beeline-hs2-connection.xml for deriving the JDBC connection URL to use when connecting to HiveServer2 from Beeline, a user can optionally add beeline-site.xml to their classpath, and within beeline-site.xml, she can specify complete JDBC URLs. A user can also specify multiple named URLs and use `beeline -c <named_url>` to connect to a specific URL. This is particularly useful when the same cluster has multiple HiveServer2 instances running with different configurations. One of the named URLs is treated as default (which is the URL that gets used when the user simply types `beeline`). An example beeline-site.xml is shown below:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
  <name>beeline.hs2.jdbc.url.tcpUrl</name>
  <value>jdbc:hive2://localhost:10000/default;user=hive;password=hive</value>
</property>

<property>
  <name>beeline.hs2.jdbc.url.httpUrl</name>
  <value>jdbc:hive2://localhost:10000/default;user=hive;password=hive;transportMode=http;httpPath=cliservice<
/value>
</property>

<property>
  <name>beeline.hs2.jdbc.url.default</name>
  <value>tcpUrl</value>
</property>
</configuration>
```

In the above example, simply typing `beeline` opens a new JDBC connection to `jdbc:hive2://localhost:10000/default;user=hive;password=hive`. If both beeline-site.xml and beeline-hs2-connection.xml are present in the classpath, the final URL is created by applying the properties specified in beeline-hs2-connection.xml on top of the URL properties derived from beeline-site.xml. As an example consider the following beeline-hs2-connection.xml:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
  <name>beeline.hs2.connection.user</name>
  <value>hive</value>
</property>
<property>
  <name>beeline.hs2.connection.password</name>
  <value>hive</value>
</property>
</configuration>
```

Consider the following beeline-site.xml:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
  <name>beeline.hs2.jdbc.url.tcpUrl</name>
  <value>jdbc:hive2://localhost:10000/default</value>
</property>

<property>
  <name>beeline.hs2.jdbc.url.httpUrl</name>
  <value>jdbc:hive2://localhost:10000/default;transportMode=http;httpPath=cliservice</value>
</property>

<property>
  <name>beeline.hs2.jdbc.url.default</name>
  <value>tcpUrl</value>
</property>
</configuration>
```

In the above example, simply typing `beeline` opens a new JDBC connection to `jdbc:hive2://localhost:10000/default;user=hive;password=hive`. When the user types `beeline -c httpUrl`, a connection is opened to `jdbc:hive2://localhost:10000/default;transportMode=http;httpPath=cliservice;user=hive;password=hive`.

Using JDBC

You can use JDBC to access data stored in a relational database or other tabular format.

1. Load the HiveServer2 JDBC driver. As of 1.2.0 applications no longer need to explicitly load JDBC drivers using Class.forName().

   For example:

   ```
   Class.forName("org.apache.hive.jdbc.HiveDriver");
   ```

2. Connect to the database by creating a `Connection` object with the JDBC driver.

   For example:

   ```
   Connection cnct = DriverManager.getConnection("jdbc:hive2://<host>:<port>", "<user>", "<password>");
   ```

   The default `<port>` is 10000. In non-secure configurations, specify a `<user>` for the query to run as. The `<password>` field value is ignored in non-secure mode.

   ```
   Connection cnct = DriverManager.getConnection("jdbc:hive2://<host>:<port>", "<user>", "");
   ```

In Kerberos secure mode, the user information is based on the Kerberos credentials.

3. Submit SQL to the database by creating a `Statement` object and using its `executeQuery()` method.

   For example:

   ```
   Statement stmt = cnct.createStatement();
   ResultSet rset = stmt.executeQuery("SELECT foo FROM bar");
   ```

4. Process the result set, if necessary.

These steps are illustrated in the sample code below.

JDBC Client Sample Code

```
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.DriverManager;

public class HiveJdbcClient {
  private static String driverName = "org.apache.hive.jdbc.HiveDriver";

  /**
   * @param args
   * @throws SQLException
   */
  public static void main(String[] args) throws SQLException {
      try {
      Class.forName(driverName);
    } catch (ClassNotFoundException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
      System.exit(1);
    }
    //replace "hive" here with the name of the user the queries should run as
    Connection con = DriverManager.getConnection("jdbc:hive2://localhost:10000/default", "hive", "");
    Statement stmt = con.createStatement();
    String tableName = "testHiveDriverTable";
    stmt.execute("drop table if exists " + tableName);
    stmt.execute("create table " + tableName + " (key int, value string)");
    // show tables
    String sql = "show tables '" + tableName + "'";
    System.out.println("Running: " + sql);
    ResultSet res = stmt.executeQuery(sql);
    if (res.next()) {
      System.out.println(res.getString(1));
    }
       // describe table
    sql = "describe " + tableName;
    System.out.println("Running: " + sql);
    res = stmt.executeQuery(sql);
    while (res.next()) {
      System.out.println(res.getString(1) + "\t" + res.getString(2));
    }

    // load data into table
    // NOTE: filepath has to be local to the hive server
    // NOTE: /tmp/a.txt is a ctrl-A separated file with two fields per line
    String filepath = "/tmp/a.txt";
    sql = "load data local inpath '" + filepath + "' into table " + tableName;
    System.out.println("Running: " + sql);
    stmt.execute(sql);

    // select * query
    sql = "select * from " + tableName;
    System.out.println("Running: " + sql);
    res = stmt.executeQuery(sql);
    while (res.next()) {
      System.out.println(String.valueOf(res.getInt(1)) + "\t" + res.getString(2));
    }

    // regular hive query
    sql = "select count(1) from " + tableName;
    System.out.println("Running: " + sql);
    res = stmt.executeQuery(sql);
    while (res.next()) {
      System.out.println(res.getString(1));
    }
  }
}
```

Running the JDBC Sample Code

```
# Then on the command-line
$ javac HiveJdbcClient.java

# To run the program using remote hiveserver in non-kerberos mode, we need the following jars in the classpath
# from hive/build/dist/lib
#     hive-jdbc*.jar
#     hive-service*.jar
#     libfb303-0.9.0.jar
#            libthrift-0.9.0.jar
#           log4j-1.2.16.jar
#           slf4j-api-1.6.1.jar
#           slf4j-log4j12-1.6.1.jar
#           commons-logging-1.0.4.jar
#
#
# To run the program using kerberos secure mode, we need the following jars in the classpath
#     hive-exec*.jar
#     commons-configuration-1.6.jar (This is not needed with Hadoop 2.6.x and later).
#  and from hadoop
#     hadoop-core*.jar (use hadoop-common*.jar for Hadoop 2.x)
#
# To run the program in embedded mode, we need the following additional jars in the classpath
# from hive/build/dist/lib
#     hive-exec*.jar
#     hive-metastore*.jar
#     antlr-runtime-3.0.1.jar
#     derby.jar
#     jdo2-api-2.1.jar
#     jpox-core-1.2.2.jar
#     jpox-rdbms-1.2.2.jar
# and from hadoop/build
#     hadoop-core*.jar
# as well as hive/build/dist/conf, any HIVE_AUX_JARS_PATH set,
# and hadoop jars necessary to run MR jobs (eg lzo codec)

$ java -cp $CLASSPATH HiveJdbcClient
```

Alternatively, you can run the following bash script, which will seed the data file and build your classpath before invoking the client. The script adds all the additional jars needed for using HiveServer2 in embedded mode as well.

```
#!/bin/bash
HADOOP_HOME=/your/path/to/hadoop
HIVE_HOME=/your/path/to/hive

echo -e '1\x01foo' > /tmp/a.txt
echo -e '2\x01bar' >> /tmp/a.txt

HADOOP_CORE=$(ls $HADOOP_HOME/hadoop-core*.jar)
CLASSPATH=.:$HIVE_HOME/conf:$(hadoop classpath)

for i in ${HIVE_HOME}/lib/*.jar ; do
    CLASSPATH=$CLASSPATH:$i
done

java -cp $CLASSPATH HiveJdbcClient
```

JDBC Data Types

The following table lists the data types implemented for HiveServer2 JDBC.

Hive Type	Java Type	Specification
TINYINT	byte	signed or unsigned 1-byte integer
SMALLINT	short	signed 2-byte integer

INT	int	signed 4-byte integer
BIGINT	long	signed 8-byte integer
FLOAT	double	single-precision number (approximately 7 digits)
DOUBLE	double	double-precision number (approximately 15 digits)
DECIMAL	java.math.BigDecimal	fixed-precision decimal value
BOOLEAN	boolean	a single bit (0 or 1)
STRING	String	character string or variable-length character string
TIMESTAMP	java.sql.Timestamp	date and time value
BINARY	String	binary data
Complex Types		
ARRAY	String – json encoded	values of one data type
MAP	String – json encoded	key-value pairs
STRUCT	String – json encoded	structured values

JDBC Client Setup for a Secure Cluster

When connecting to HiveServer2 with Kerberos authentication, the URL format is:

```
jdbc:hive2://<host>:<port>/<db>;principal=<Server_Principal_of_HiveServer2>
```

The client needs to have a valid Kerberos ticket in the ticket cache before connecting.

NOTE: If you don't have a "/" after the port number, the jdbc driver does not parse the hostname and ends up running HS2 in embedded mode . So if you are specifying a hostname, make sure you have a "/" or "/<dbname>" after the port number.

In the case of LDAP, CUSTOM or PAM authentication, the client needs to pass a valid user name and password to the JDBC connection API.

```
To use sasl.qop, add the following to the sessionconf part of your Hive JDBC hive connection string, e.g.

jdbc:hive://hostname/dbname;sasl.qop=auth-int
```

For more information, see Setting Up HiveServer2.

Multi-User Scenarios and Programmatic Login to Kerberos KDC

In the current approach of using Kerberos you need to have a valid Kerberos ticket in the ticket cache before connecting. This entails a static login (using kinit, key tab or ticketcache) and the restriction of one Kerberos user per client. These restrictions limit the usage in middleware systems and other multi-user scenarios, and in scenarios where the client wants to login programmatically to Kerberos KDC.

One way to mitigate the problem of multi-user scenarios is with secure proxy users (see HIVE-5155). Starting in Hive 0.13.0, support for secure proxy users has two components:

- Direct proxy access for privileged Hadoop users (HIVE-5155). This enables a privileged user to directly specify an alternate session user during the connection. If the connecting user has Hadoop level privilege to impersonate the requested userid, then HiveServer2 will run the session as that requested user.
- Delegation token based connection for Oozie (OOZIE-1457). This is the common mechanism for Hadoop ecosystem components.
  Proxy user privileges in the Hadoop ecosystem are associated with both user names and hosts. That is, the privilege is available for certain users from certain hosts. Delegation tokens in Hive are meant to be used if you are connecting from one authorized (blessed) machine and later you need to make a connection from another non-blessed machine. You get the delegation token from a blessed machine and connect using the delegation token from a non-blessed machine. The primary use case is Oozie, which gets a delegation token from the server machine and then gets another connection from a Hadoop task node.

  If you are only making a JDBC connection as a privileged user from a single blessed machine, then direct proxy access is the simpler approach. You can just pass the user you need to impersonate in the JDBC URL by using the hive.server2.proxy.user=<*user*> parameter.

  See examples in ProxyAuthTest.java.

  Support for delegation tokens with HiveServer2 binary transport mode hive.server2.transport.mode has been available starting 0.13.0; support for this feature with HTTP transport mode was added in HIVE-13169, which should be part of Hive 2.1.0.

The other way is to use a pre-authenticated Kerberos Subject (see HIVE-6486). In this method, starting with Hive 0.13.0 the Hive JDBC client can use a pre-authenticated subject to authenticate to HiveServer2. This enables a middleware system to run queries as the user running the client.

Using Kerberos with a Pre-Authenticated Subject

To use a pre-authenticated subject you will need the following changes.

1. Add hive-exec*.jar to the classpath in addition to the regular Hive JDBC jars (commons-configuration-1.6.jar and hadoop-core*.jar are not required).
2. Add auth=kerberos and kerberosAuthType=fromSubject JDBC URL properties in addition to having the “principal" url property.
3. Open the connection in Subject.doAs().

The following code snippet illustrates the usage (refer to HIVE-6486 for a complete test case):

```
static Connection getConnection( Subject signedOnUserSubject ) throws Exception{
       Connection conn = (Connection) Subject.doAs(signedOnUserSubject, new PrivilegedExceptionAction<Object>()
           {
               public Object run()
               {
                       Connection con = null;
                       String JDBC_DB_URL = "jdbc:hive2://HiveHost:10000/default;" ||
                                            "principal=hive/localhost.localdomain@EXAMPLE.COM;" ||
                                            "kerberosAuthType=fromSubject";
                       try {
                               Class.forName(JDBC_DRIVER);
                               con =  DriverManager.getConnection(JDBC_DB_URL);
                       } catch (SQLException e) {
                               e.printStackTrace();
                       } catch (ClassNotFoundException e) {
                               e.printStackTrace();
                       }
                       return con;
               }
           });
       return conn;
}
```

JDBC Fetch Size

Gives the JDBC driver a hint as to the number of rows that should be fetched from the database when more rows are needed by the client. The default value, used for every statement, can be specified through the JDBC connection string. This default value may subsequently be overwritten, per statement, with the JDBC API. If no value is specified within the JDBC connection string, then the default fetch size is retrieved from the HiveServer2 instance as part of the session initiation operation.

`jdbc:hive2://<host>:<port>/<db>;fetchsize=<value>`

Hive Version 4.0

The Hive JDBC driver will receive a preferred fetch size from the instance of HiveServer2 it has connected to. This value is specified on the server by the *hive.server2.thrift.resultset.default.fetch.size* configuration.

The JDBC fetch size is only a hint and the server will attempt to respect the client's requested fetch size though with some limits. HiveServer2 will cap all requests at a maximum value specified by the *hive.server2.thrift.resultset.max.fetch.size* configuration value regardless of the client's requested fetch size.

While a larger fetch size may limit the number of round-trips between the client and server, it does so at the expense of additional memory requirements on the client and server.

The default JDBC fetch size value may be overwritten, per statement, with the JDBC API:

- Setting a value of 0 instructs the driver to use the fetch size value preferred by the server
- Setting a value greater than zero will instruct the driver to fetch that many rows, though the actual number of rows returned may be capped by the server
- If no fetch size value is explicitly set on the JDBC driver's statement then the driver's default value is used
  - If the fetch size value is specified within the JDBC connection string, this is the default value
  - If the fetch size value is absent from the JDBC connection string, the server's preferred fetch size is used as the default value

Python Client

A Python client driver is available on github. For installation instructions, see Setting Up HiveServer2: Python Client Driver.

Ruby Client

A Ruby client driver is available on github at https://github.com/forward3d/rbhive.

Integration with SQuirrel SQL Client

1. Download, install and start the SQuirrel SQL Client from the SQuirrel SQL website.
2. Select 'Drivers -> New Driver...' to register Hive's JDBC driver that works with HiveServer2.
    a. Enter the driver name and example URL:

```
Name: Hive
Example URL: jdbc:hive2://localhost:10000/default
```

3. Select 'Extra Class Path -> Add' to add the following jars from your local Hive and Hadoop distribution.

```
HIVE_HOME/lib/hive-jdbc-*-standalone.jar
HADOOP_HOME/share/hadoop/common/hadoop-common-*.jar
```

> **Version information**
>
> Hive JDBC standalone jars are used in Hive 0.14.0 onward (HIVE-538); for previous versions of Hive, use `HIVE_HOME/build/dist/lib/*.jar` instead.
>
> The hadoop-common jars are for Hadoop 2.0; for previous versions of Hadoop, use `HADOOP_HOME/hadoop-*-core.jar` instead.

4. Select 'List Drivers'. This will cause SQuirrel to parse your jars for JDBC drivers and might take a few seconds. From the 'Class Name' input box select the Hive driver for working with HiveServer2:

```
org.apache.hive.jdbc.HiveDriver
```

5. Click 'OK' to complete the driver registration.
6. Select 'Aliases -> Add Alias...' to create a connection alias to your HiveServer2 instance.
    a. Give the connection alias a name in the 'Name' input box.
    b. Select the Hive driver from the 'Driver' drop-down.
    c. Modify the example URL as needed to point to your HiveServer2 instance.
    d. Enter 'User Name' and 'Password' and click 'OK' to save the connection alias.
    e. To connect to HiveServer2, double-click the Hive alias and click 'Connect'.

When the connection is established you will see errors in the log console and might get a warning that the driver is not JDBC 3.0 compatible. These alerts are due to yet-to-be-implemented parts of the JDBC metadata API and can safely be ignored. To test the connection enter *SHOW TABLES* in the console and click the run icon.

Also note that when a query is running, support for the 'Cancel' button is not yet available.

Integration with SQL Developer

Integration with Oracle SQLDeveloper is available using JDBC connection.

https://community.hortonworks.com/articles/1887/connect-oracle-sql-developer-to-hive.html

Integration with DbVisSoftware's DbVisualizer

1. Download, install and start DbVisualizer free or purchase DbVisualizer Pro from https://www.dbvis.com/.
2. Follow instructions on github.

Advanced Features for Integration with Other Tools

Supporting Cookie Replay in HTTP Mode

Version 1.2.0 and later

This option is available starting in Hive 1.2.0.

HIVE-9709 introduced support for the JDBC driver to enable cookie replay. This is turned on by default so that incoming cookies can be sent back to the server for authentication.

The JDBC connection URL when enabled should look like this:

```
jdbc:hive2://<host>:<port>/<db>?transportMode=http;httpPath=<http_endpoint>;cookieAuth=true;
cookieName=<cookie_name>
```

- cookieAuth is set to `true` by default.
- cookieName: If any of the incoming cookies' keys match the value of cookieName, the JDBC driver will not send any login credentials/Kerberos ticket to the server. The client will just send the cookie alone back to the server for authentication. The default value of cookieName is hive.server2.auth (this is the HiveServer2 cookie name).
- To turn off cookie replay, cookieAuth=false must be used in the JDBC URL.
- **Important Note:** As part of HIVE-9709, we upgraded Apache http-client and http-core components of Hive to 4.4. To avoid any collision between this upgraded version of HttpComponents and other any versions that might be present in your system (such as the one provided by Apache Hadoop 2.6 which uses http-client and http-core components version of 4.2.5), the client is expected to set CLASSPATH in such a way that Beeline-related jars appear before HADOOP lib jars. This is achieved via setting HADOOP_USER_CLASSPATH_FIRST=true before using hive-jdbc. In fact, in bin/beeline.sh we do this!

Using 2-way SSL in HTTP Mode

Version 1.2.0 and later

This option is available starting in Hive 1.2.0.

HIVE-10447 enabled the JDBC driver to support 2-way SSL in HTTP mode. Please note that HiveServer2 currently does not support 2-way SSL. So this feature is handy when there is an intermediate server such as Knox which requires client to support 2-way SSL.

JDBC connection URL:

```
jdbc:hive2://<host>:<port>/<db>;ssl=true;twoWay=true;sslTrustStore=<trust_store_path>;
trustStorePassword=<trust_store_password>;sslKeyStore=<key_store_path>;keyStorePassword=<key_store_password>;
transportMode=http;httpPath=<http_endpoint>
```

- *<trust_store_path>* is the path where the client's truststore file lives. This is a mandatory non-empty field.
- *<trust_store_password>* is the password to access the truststore.
- *<key_store_path>* is the path where the client's keystore file lives. This is a mandatory non-empty field.
- *<key_store_password>* is the password to access the keystore.

For versions earlier than 0.14, see the version note above.

In the environment where exposing `trustStorePassword` and `keyStorePassword` in the connection URL is a security concern, a new option `storePasswordPath` is introduced with HIVE-27308 that can be used in URL instead of `trustStorePassword` and `keyStorePassword`. `storePasswordPath` value hold the path to the local keystore file storing the `trustStorePassword` and `keyStorePassword` aliases. When the existing `trustStorePassword` or `keyStorePassword` is present in URL along with `storePasswordPath`, respective password is directly obtained from password option. Otherwise, fetches the particular alias from local keystore file(i.e., existing password options are preferred over `storePasswordPath`).

JDBC connection URL with `storePasswordPath`:

```
jdbc:hive2://<host>:<port>/<db>;ssl=true;twoWay=true;sslTrustStore=<trust_store_path>;
sslKeyStore=<key_store_path>;storePasswordPath=store_password_path>;transportMode=http;
httpPath=<http_endpoint>
```

A local keystore file can be created leveraging hadoop credential command with `trustStorePassword` and `keyStorePassword` aliases like below. And this file can be passed with `storePasswordPath` option in the connection URL.

hadoop credential create **trustStorePassword** -value *mytruststorepassword* -provider *localjceks://file/tmp/client_creds.jceks*

hadoop credential create **keyStorePassword** -value *mykeystorepassword* -provider *localjceks://file/tmp/client_creds.jceks*

Passing HTTP Header Key/Value Pairs via JDBC Driver

Version 1.2.0 and later

This option is available starting in Hive 1.2.0.

HIVE-10339 introduced an option for clients to provide custom HTTP headers that can be sent to the underlying server (Hive 1.2.0 and later).

JDBC connection URL:

```
jdbc:hive2://<host>:<port>/<db>;transportMode=http;httpPath=<http_endpoint>;http.header.<name1>=<value1>;
http.header.<name2>=<value2>
```

When the above URL is specified, Beeline will call underlying requests to add an HTTP header set to <*name1*> and <*value1*> and another HTTP header set to <*name2*> and <*value2*>. This is helpful when the end user needs to send identity in an HTTP header down to intermediate servers such as Knox via Beeline for authentication, for example `http.header.USERNAME=<value1>;http.header.PASSWORD=<value2>`.

For versions earlier than 0.14, see the version note above.

Passing Custom HTTP Cookie Key/Value Pairs via JDBC Driver

In Hive version 3.0.0 HIVE-18447 introduced an option for clients to provide custom HTTP cookies that can be sent to the underlying server. Some authentication mechanisms, like Single Sign On, need the ability to pass a cookie to some intermediate authentication service like Knox via the JDBC driver.

JDBC connection URL:

```
jdbc:hive2://<host>:<port>/<db>;transportMode=http;httpPath=<http_endpoint>;http.cookie.<name1>=<value1>;
http.cookie.<name2>=<value2>
```

When the above URL is specified, Beeline will call underlying requests to add HTTP cookie in the request header, and will set it to <*name1*>=<*value1*> and <*name2*>=<*value2*>.