DataStax Bulk Loader 1.2.0 (Latest version)

Updated: 2019-01-19-08:00

Table of Contents

About DataStax Bulk Loader - dsbulk

The DataStax Bulk Loader tool `dsbulk` can be used to load and unload DSE data in either CSV or JSON format.

About this document

Welcome to the DataStax Bulk Loader documentation. To ensure that you get the best experience in using this document, take a moment to look at the Tips for using DataStax documentation.

About DataStax Bulk Loader - dsbulk

The DataStax Bulk Loader tool is designed to provide users with the ability to both load and unload data in and out of DSE efficiently and reliably. `dsbulk` efficiently loads small or large amounts of data, and supports both developer and production environments. Using `dsbulk`, CSV or JSON files can be rapidly loaded or unloaded to or from DSE. CSV or JSON files from both relational database exports and original data may be inserted into the DSE transactional database. The tool is supported for both Linux and Windows platforms.

Features in DataStax Bulk Loader

- Can migrate data into DSE from another DSE or Apache Cassandra™ cluster
  - # Can unload data from any Cassandra 2.1 or later data source
  - # Can load data to DSE 5.0 or later
- CSV and JSON are supported formats
- Files, directories, stdin/stdout, and web URLs can be used for either source or destination
- Performance improvements of 2-3 times faster compared to cqlsh COPY, due to multi-threaded operation
- Secure authentication via Kerberos or username/password over SSL
- Configurable data parsing (for instance, date formatting is configurable)
- Performance and progress reporting
- Command line tool for both Linux and Windows:
  - # Can use configuration files to simplify command line calls to dsbulk
  - # Tunable parameters to optimize loading and unloading times
  - # Enhancements allow secure connections for loading and unloading data

See the DataStax blog post Introducing the DataStax Bulk Loader.

DataStax Bulk Loader release notes

Release notes for DataStax Bulk Loader.

DataStax Bulk Loader can migrate data in CSV or JSON format into DSE from another DSE or Apache Cassandra™ cluster.

- Can unload data from any Cassandra 2.1 or later data source
- Can load data to DSE 5.0 or later

For DataStax Enterprise release notes:

- DSE 6.7 release notes
- DSE 6.0 release notes
- DSE 5.1 release notes

DataStax Bulk Loader 1.2.0 release notes

1 August 2018

DataStax Bulk Loader 1.2.0 release notes include:

- 1.2.0 Changes and enhancements *(page 5)*
- 1.2.0 Resolved issues *(page 5)*

1.2.0 Changes and enhancements

After upgrade to 1.2.0, be sure to review and adjust scripts to use changed settings.

- Improve range split algorithm in multi-DC and vnodes environments. (DAT-252)
- Support simplified notation for JSON arrays and objects in collection fields. (DAT-317)

1.2.0 Resolved issues:

- CSVWriter trims leading/trailing whitespace in values. (DAT-302)
- CSV connector fails when the number of columns in a record is greater than 512. (DAT-311)
- Bulk Loader fails when mapping contains a primary key column mapped to a function. (DAT-326)

DataStax Bulk Loader 1.1.0 release notes

18 June 2018

DataStax Bulk Loader 1.1.0 release notes include:

- 1.1.0 Changes and enhancements *(page 6)*
- 1.1.0 Resolved issues *(page 6)*

1.1.0 Changes and enhancements

After upgrade to 1.1.0, be sure to review and adjust scripts to use changed settings.

- Combine batch.mode and batch.enabled into a single setting: batch.mode. If you are using the batch.enabled setting in scripts, change to batch.mode with value DISABLED. (DAT-287)
- Improve handling of Univocity exceptions. (DAT-286)
- Logging improvements. (DAT-290)
  # Log messages are logged only to `operation.log`. Logging does not print to stdout.
  # Configurable logging levels with the log.verbosity setting.
  # The setting log.ansiEnabled is changed to log.ansiMode.
- New count workflow. (DAT-291, DAT-299)
  # Supports counting rows in a table.
  # Configurable counting mode.
  # When mode = partitions, configurable number of partitions to count. Support to count the number of rows for the *N* biggest partitions in a table.
- Counter tables are supported for load and unload. (DAT-292)
- Improve validation to include user-supplied queries and mappings. (DAT-294)
- The codec.timestamp CQL_DATE_TIME setting is renamed to CQL_TIMESTAMP. Adjust scripts to use the new setting. (DAT-298)

1.1.0 Resolved issues:

- Generated query does not contain all token ranges when a range wraps around the ring. (DAT-295)
- Empty map values do not work when loading using dsbulk. (DAT-297)
- DSBulk cannot handle columns of type list<timestamp>. (DAT-288)
- Generated queries do not respect indexed mapping order. (DAT-289)
- DSBulk fails to start with Java 10+. (DAT-300)

DataStax Bulk Loader 1.0.2 release notes

5 June 2018

DataStax Bulk Loader 1.0.2 release notes include:

- 1.0.2 Changes and enhancements *(page 6)*

1.0.2 Changes and enhancements

- DataStax Bulk Loader 1.0.2 is bundled with DSE 6.0.1. (DSP-16206)

- Configure whether to use ANSI colors and other escape sequences in log messages *(page 42)* printed to standard output and standard error. (DAT-249)

DataStax Bulk Loader 1.0.1 release notes

17 April 2018

DataStax Bulk Loader 1.0.1 release notes include:

1.0.1 Changes and enhancements

- DataStax Bulk Loader (dsbulk *(page 4)*) version 1.0.1 is automatically installed with DataStax Enterprise, and can also be installed as a standalone tool. DataStax Bulk Loader 1.0.1 is supported for use with DSE 5.0 and later. (DSP-13999, DSP-15623)
- Support to manage special characters on the command line *(page 14)* and in the configuration file. (DAT-229)
- Improve error messages for incorrect mapping. (DAT-235)
- Improved monitoring options *(page 44)*. (DAT-238)
- Detect console width on Windows. (DAT-240)
- Null words are supported by all connectors. The schema.nullStrings is changed to codec.nullWords. Renamed the convertTo and convertFrom methods. See Codec options *(page 28)* and Schema options *(page 24)*. (DAT-241)
- Use Logback to improve filtering to make stack traces more readable and useful. On ANSI-compatible terminals, the date prints in green, the hour in cyan, the level is blue (INFO) or red (WARN), and the message prints in black. (DAT-242)
- Improved messaging for completion with errors. (DAT-243)
- Settings schema.allowExtraFields and schema.allowMissingFields are added to reference.conf *(page 24)*. (DAT-244)
- Support is dropped for using `:port` to specify the port to connect to. Specify the port for all hosts only with driver.port *(page 19)*. (DAT-245)

1.0.1 Resolved issues

- Numeric overflows should display the original input that caused the overflow. (DAT-237)
- Null words are not supported by all connectors. (DAT-241)
- Addresses might not be properly translated when cluster has custom native port. (DAT-245)

Installing DataStax Bulk Loader

Steps for installing DataStax Bulk Loader (dsbulk).

See Installing DataStax Bulk Loader.

Architecture

The DataStax Workflow Engine is the component responsible for the orchestration of loading and unloading operations. The main features are:

- Configuration: The engine collects user-supplied settings, merges them with default values and configures the loading/unloading operation to run.
- Connection: The engine handles the driver connection to DSE and manages driver-specific settings, as well as supports authentication and SSL encryption.
- Conversion: The engine handles data type conversions, e.g. boolean, number, date conversions from anything (typically, strings or raw bytes as emitted by a connector) to appropriate internal representations (typically, Java Temporal or Number objects). It also handles `NULL` and `UNSET` values.
- Mapping: The engine analyzes metadata gathered from the driver and infers the appropriate `INSERT` or `SELECT` prepared statement, then checks this information against user-supplied information about the data source, to infer the bound variables to use.
- Monitoring: The engine reports metrics about all its internal components, mainly the connector and the bulk executor.
- Error Handling: The engine handles errors from both connectors and the bulk executor, and reports read, parse, and write failures. These are redirected to a configurable "bad file" that contains sources that could not be loaded.

Figure 1: Loading Workflow

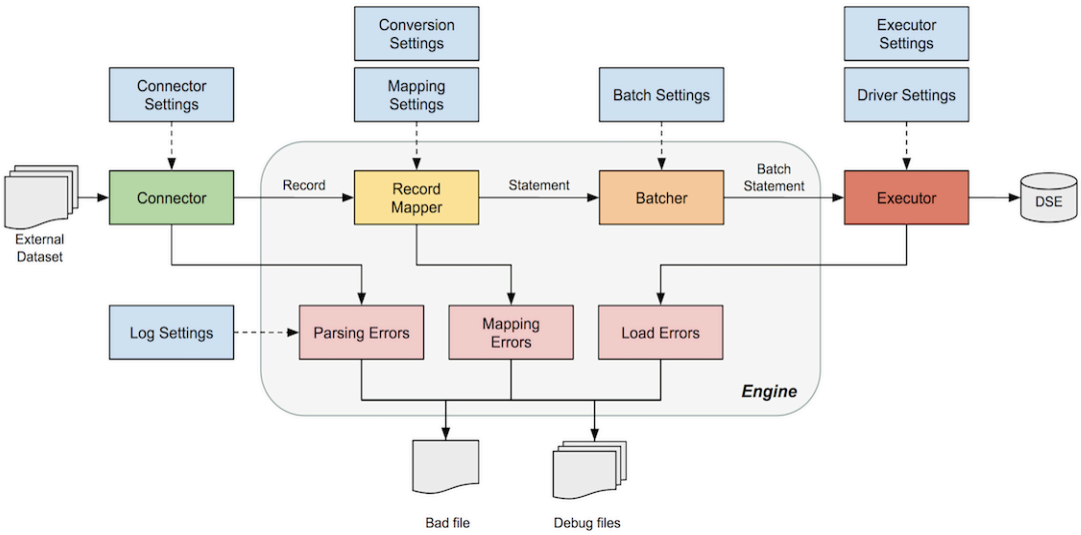
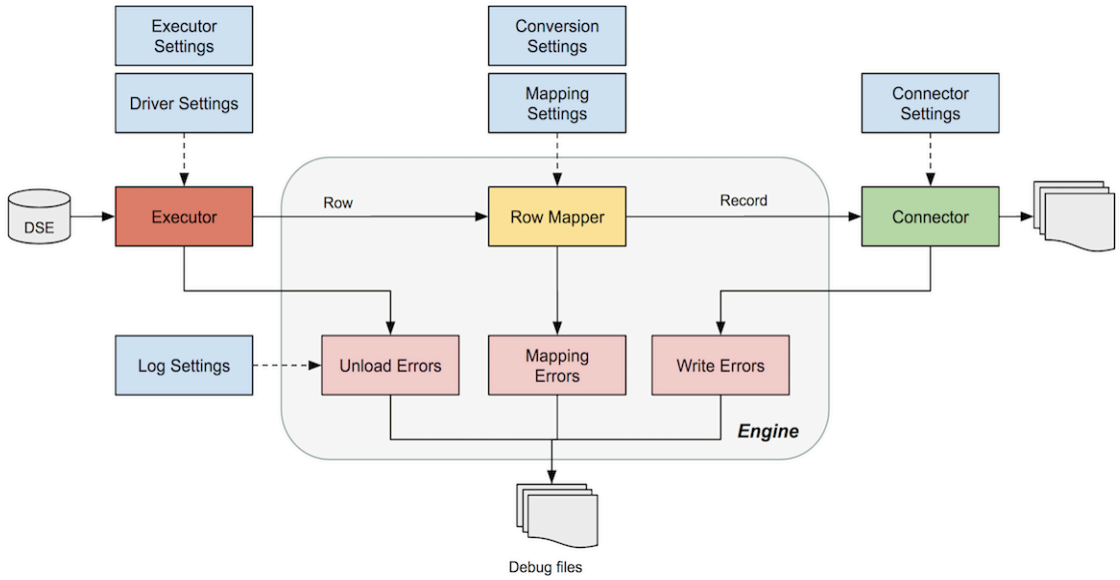


Figure 2: Unloading Workflow



Getting Started

This guide demonstrates the key features of using `dsbulk` to get a user started.

Prerequisites:

Obtain the following information and resources:

- Your DataStax Academy registration email address and Downloads Key or Profile Name and password.
- Download and install `dsbulk`.
- Create DataStax tables using a tool like CQLSH. `dsbulk` loads data, but it does not create keyspaces or tables.

Key features

- Simple loading *(page 11)* without configuration file
- Simple unloading *(page 11)* without configuration file
- Creating a configuration file *(page 12)* for use with loading or unloading
- Using SSL *(page 12)* with `dsbulk`

Simple loading without configuration file

1. Loading CSV data with a `dsbulk load` command:

   Specify two hosts (initial contact points) that belong to the desired cluster and load from a local file `export.csv` with headers into keyspace *ks1* and table *table1*:

   ```
   $ dsbulk load -url export.csv -k ks1 -t table1 -h '10.200.1.3,
    10.200.1.4' -header true
   ```

   `url` can designate the path to a resource, such as a local file, or a web URL from which to read/write data.

   Specify an external source of data, as well as a port for the cluster hosts:

   ```
   dsbulk load -url https://svr/data/export.csv -k ks1 -t table1 -h
    '10.200.1.3, 10.200.1.4' -port 9876
   ```

   Load CSV data from `stdin` as it is generated from a loading script *generate_data*. The data is loaded to the keyspace *ks1* and table *table1* in a cluster with a `localhost` contact point (default if no hosts are defined). By default if not specified, the field names are read from a header row in the input file.

   ```
   generate_data | dsbulk load -url stdin:/ -k ks1 -t table1
   ```

Simple unloading without configuration file

2. Unloading CSV data with a `dsbulk unload` command:

Specify the external file to write the data to from keyspace *ks1* and table *table1*:

```
dsbulk unload -url myData.csv -k ks1 -t table1
```

Creating a configuration file

3. The configuration file for setting values for `dsbulk` are written in a simple format, one option per line:

```
############ MyConfFile.conf ############

dsbulk {
   # The name of the connector to use
   connector.name = "csv"
   # CSV field delimiter
   connector.csv.delimiter = "|"
   # The keyspace to connect to
   schema.keyspace = "myKeyspace"
   # The table to connect to
   schema.table = "myTable"
   # The field-to-column mapping
   schema.mapping = "0=name, 1=age, 2=email"
}
```

   Tip: Settings in the config file always start with the `dsbulk` prefix, while on the command line, this prefix must be omitted. To avoid confusion, configuration files are formatted with the following equivalent HOCON syntax: `dsbulk { connector.name = "csv" ... }`.

   To use the configuration file, specify `-f filename`, where `filename` is the configuration file:

```
dsbulk load -f myConfFile.conf -url export.csv -k ks1 -t table1
```

Using SSL with dsbulk

4. To use SSL with `dsbulk`, first refer to DSE Security docs to set up SSL. The SSL options can be specified on the command line, but a configuration file is a good option given the long option names:

```
driver.ssl.keystore.password = cassandra
driver.ssl.keystore.path = "/Users/johndoe/tmp/ssl/keystore.node0"
driver.ssl.provider = OpenSSL
driver.ssl.truststore.password = dserocks
driver.ssl.truststore.path = "/Users/johndoe/tmp/ssl/truststore.node0"
```

   The command is:

```
dsbulk load -f mySSLFile.conf -url file1.csv -k ks1 -t table1
```

DataStax Bulk Loader reference

dsbulk

The DataStax data loader `dsbulk` can be used for both loading data from a variety of sources and unloading data from DataStax Enterprise for transfer, use, or storage of data. Two subcommands, `load` and `unload`, are straightforward. Both subcommands require either the options `keyspace` and `table` or a `schema.query`, plus a data source *(page 21)*. A wide variety of options are also available to help users tailor how DSE data loader operates. These options have defined default values or values inferred from the input data, if the operation is loading, or from the database data, if the operation is unloading. The options described here are grouped functionally, so that additional requirements can be noted. For example, if loading or unloading CSV data, the `connector.csv.url` option must be set, specifying the path or URL of the CSV data file used for loading or unloading.

The standalone tool is launched using the command `dsbulk` from within the `bin` directory of your distribution. The tool also provides inline help for all settings. A configuration file specifying option values can be used, or options can be specified on the command line. Options specified on the command line will override the configuration file option settings.

Synopsis

```
$ dsbulk ( load | unload | count ) [options]
  (( -k | --keyspace ) keyspace_name
  ( -t | --table ) table_name)
  | ( --schema.query string )
  [ help | --help ]
```

The options can be used in short form (`-k keyspace_name`) or long form (`--schema.keyspace keyspace_name`).

Table 1: Legend

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
Italics	Variable value. Replace with a valid option or user-defined value.
`[ ]`	Optional. Square brackets (`[ ]`) surround optional command arguments. Do not type the square brackets.
`( )`	Group. Parentheses (`( )`) identify a group to choose from. Do not type the parentheses.
`|`	Or. A vertical bar (`|`) separates alternative elements. Type any one of the elements. Do not type the vertical bar.

Syntax conventions	Description
`...`	Repeatable. An ellipsis (`...`) indicates that you can repeat the syntax element as often as required.
`'Literal string'`	Single quotation (`'`) marks must surround literal strings in dsbulk statements. Use single quotation marks to preserve upper case.
`{ key:value }`	Map collection. Braces (`{ }`) enclose map collections or key value pairs. A colon separates the key and the value.
`<datatype1,datatype2>`	Set, list, map, or tuple. Angle brackets (`< >`) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
`[ -- ]`	Separate the command line options from the command arguments with two hyphens (`--`). This syntax is useful when arguments might be mistaken for command line options.

General use

Get general help about `dsbulk` and the common options:

```
$ dsbulk help
```

Get help about particular `dsbulk` options, such as `connector.csv` options using the `help` subcommand:

```
$ dsbulk help connector.csv
```

Run `dsbulk -c csv` with `--help` option to see its short options, along with the general help:

```
$ dsbulk -c csv --help
```

Escaping and Quoting Command Line Arguments

When supplied via the command line, all option values are expected to be in valid HOCON syntax. For example, control characters, the backslash character, and the double-quote character all need to be properly escaped. For example, `\t` is the escape sequence that corresponds to the tab character, whereas `\\` is the escape sequence for the backslash character:

```
dsbulk load -delim '\t' -url 'C:\\Users\\My Folder'
```

In general, string values containing special characters also need to be properly quoted with double-quotes, as required by the HOCON syntax:

```
dsbulk load -url '"C:\\Users\\My Folder"'
```

However, when the expected type of an option is a string, it is possible to omit the surrounding double-quotes, for convenience. Thus, note the absence of the double-quotes in the first example. Similarly, when an argument is a list, it is possible to omit the surrounding square brackets; making the following two lines equivalent:

```
dsbulk load --codec.nullStrings 'NIL, NULL'
dsbulk load --codec.nullStrings '[NIL, NULL]'
```

The same applies for arguments of type map: it is possible to omit the surrounding curly braces, making the following two lines equivalent:

```
dsbulk load --connector.json.deserializationFeatures
 '{ USE_BIG_DECIMAL_FOR_FLOATS : true }'
dsbulk load --connector.json.deserializationFeatures
 'USE_BIG_DECIMAL_FOR_FLOATS : true'
```

This syntactic sugar is only available for command line arguments of type string, list or map; all other option types, as well as all options specified in a configuration file must be fully compliant with HOCON syntax, and it is the user's responsibility to ensure that such options are properly escaped and quoted.

Loading data examples

Load data from CSV data read from `stdin` to table *table1* in keyspace *ks1*:

```
$ dsbulk load -k ks1 -t table1
```

Load a configuration file from `/tmp/dsbulk_load.conf` to use for loading the file `export.csv` to table *table1* in keyspace *ks1*:

```
$ dsbulk load -f /tmp/dsbulk_load.conf --connector.csv.url export.csv -k
 ks1 -t table1
```

Load the file `export.csv` to table *table1* in keyspace *ks1* using the short form option for `url`:

```
$ dsbulk load -url export.csv -k ks1 -t table1
```

Load the file `export.csv` to table *table1* in keyspace *ks1* using the short form option for `url` and the tab character as a field delimiter:

```
$ dsbulk load -k ks1 -t table1 -url export.csv -delim '\t'
```

Load the file `/tmp/export.csv` to table *table1* in keyspace *ks1* using the long form option for `url`:

```
$ dsbulk load --connector.csv.url file:///tmp/export.csv -k ks1 -t table1
```

> **Note:** Note that `file:///tmp/export.csv` is loading from localhost, hence the empty host in the `file://` designation.

Load table *table1* in keyspace *ks1* from a gzipped CSV file by unzipping it to stdout and piping to `stdin` of dsbulk:

```
$ gzcat table1.csv.gz | dsbulk load -k ks1 -t table1
```

Specify a few hosts (initial contact points) that belong to the desired cluster and load from a local file, without headers. Map field indices of the input to table columns with `-m`:

```
$ dsbulk load -url ~/export.csv -k ks1 -t table1 -h '10.200.1.3,
 10.200.1.4' -header false -m '0=col1,1=col3'
```

Specify port 9876 for the cluster hosts and load from an external source url:

```
$ dsbulk load -url https://192.168.1.100/data/export.csv -k ks1 -t table1
 -h '10.200.1.3,10.200.1.4' -port 9876
```

Load all csv files from a directory. The files do not have a header row, `-header false`. Map field indices of the input to table columns with `-m`:

```
$ dsbulk load -url ~/export-dir -k ks1 -t table1 -header false -m
 '0=col1,1=col3'
```

With default port for cluster hosts, keyspace, table, and mapping set in `conf/application.conf`:

```
$ dsbulk load -url https://192.168.1.100/data/export.csv -h
 '10.200.1.3,10.200.1.4'
```

Load table *table1* in keyspace *ks1* from a CSV file, where double-quote characters in fields are escaped with a double-quote; for example, `"f1","value with ""quotes"" and more"` is a line in the CSV file:

```
$ dsbulk load -url ~/export.csv -k ks1 -t table1 -escape '\"'
```

Loading collections to a table has some specific helpful simplification. Collections inside a CSV file can either contain valid JSON or simpler non-compliant JSON. For example, for the following table:

```
$ CREATE TABLE t1 (col1 set<int> PRIMARY KEY, col2 list<int>, col3
 map<text,int>);
```

This pipe-delimited CSV file contains valid JSON and could be loaded:

```
$ col1|col2|col3
[1,2,3]|[1,2,3]|{"key1":1,"key2":2}
```

or this similar CSV file could also be loaded:

```
$ col1|col2|col3
1,2,3|1,2,3|"key1":1,"key2":2
```

Notice that the surrounding brackets and braces are omitted from the valid JSON in the third column.

Important: Simplification of the JSON data will not work for nested collections. If your table has a column *col4* of type `list<list<int>>`, only the outermost structure can omit the surrounding characters. For example:

```
$ col1,col2,col3,col4
[1,2,3]|[1,2,3]|{"key1":1,"key2":2}|[[1,2,3],[4,5,6]]
```

can be simplifed to:

```
$ col1,col2,col3,col4
1,2,3|1,2,3|"key1":1,"key2":2|[1,2,3],[4,5,6]
```

but the inner list items cannot be simplified further.

Unloading data examples

Unloading is the process of extracting data from DataStax Enterprise into a CSV or JSON file. Many options used in both loading and unloading are the same.

Unload data to `stdout` from the *ks1.table1* table in a cluster with a `localhost` contact point. Column names in the table map to field names in the data. Field names must be emitted in a *header row* in the output:

```
$ dsbulk unload -k ks1 -t table1
```

Unload data to `stdout` from the *ks1.table1* table and gzip the result:

```
$ dsbulk unload -k ks1 -t table1 | gzip > table1.gz
```

Unload data to a local directory (which may not yet exist):

```
$ dsbulk unload -url ~/data-export -k ks1 -t table1
```

Exit codes

The `dsbulk` command has exit codes that are returned to a calling process. The following values link the integer value returned with the status:

Common options

Some options are commonly required to use `dsbulk`. In the following list, required options are designated.

The options can be used in short form (`-k` *`keyspace_name`*) or long form (`--schema.keyspace` *`keyspace_name`*).

-f *`filename`*

Load options from the given file rather than from `dsbulk_home/conf/application.conf`.

Default: unspecified

-c,--connector.name csv | json

The name of the connector to use.
Default: csv

-k,--schema.keyspace *`string`*

Keyspace used for loading, unloading, or counting data. Keyspace names should not be quoted and are case-sensitive. `MyKeyspace` will match a keyspace named *MyKeyspace* but not *mykeyspace*. Required option if `schema.query` is not specified; otherwise, optional.
Default: unspecified

-t,--schema.table *`string`*

Table used for loading, unloading, or counting data. Table names should not be quoted and are case-sensitive. `MyTable` will match a table named *MyTable* but not *mytable*. Required option if `schema.query` is not specified; otherwise, optional.
Default: unspecified

-url,--connector.(csv|json).url *`string`*

The URL or path of the resource(s) to read from or wrote to. Possible options are `-` (representing stdin for reading and stdout for writing) and `file` (filepath). File URLs can also be expressed as simple paths without the `file` prefix. A directory of files can also be specified.
Default: -

-delim,--connector.csv.delimiter *`string`*

The character to use as field delimiter.
Default: , (a comma)

-header,--connector.csv.header (true | false)

Enable or disable whether the files to read or write begin with a header line. If enabled for loading, the first non-empty line in every file will assign field names for each record column, in lieu of schema.mapping *(page 25)*, `fieldA = col1, fieldB = col2, fieldC = col3`. If disabled for loading, records will not contain fields names, only field indexes, `0 = col1, 1 = col2, 2 = col3`. For unloading, if this setting is enabled, each file will begin with a header line, and if disabled, each file will not contain a header line.

Note: This option will apply to all files loaded or unloaded.

Default: true

-h,--driver.hosts *`host_name(s)`*

The contact points to use for the initial connection to the cluster. This must be a comma-separated list of hosts, each specified by a host-name or IP address. If the host is a DNS name that resolves to multiple A-records, all the corresponding addresses will be used. Do not use `localhost` as a host-name (since it resolves to

both IPv4 and IPv6 addresses on some platforms). The port for all hosts must be specified with `driver.port`.
Default: 127.0.0.1

-port,--driver.port *`port_number`*

The port to connect to at initial contact points. Note that all nodes in a cluster must accept connections on the same port number.
Default: 9042

Connector options

Connectors allow different types of data to be loaded and unloaded using `dsbulk`. The general format for connector options is:

```
--connector.name.option ( string | number )
```

. An example is `connector.csv.url`.

Which URL protocols are available depend on which URL stream handlers have been installed, but at least the **file** protocol is guaranteed to be supported for reading and writing, and the `HTTP/HTTPS` protocols are guaranteed to be supported for reading.

The file protocol can be used with all supported file systems, local or not.

- When reading: the URL can point to a single file, or to an existing directory; in case of a directory, the *fileNamePattern* setting can be used to filter files to read, and the *recursive* setting can be used to control whether or not the connector should look for files in subdirectories as well.
- When writing: the URL will be treated as a directory; if it doesn't exist, the loader will attempt to create it; CSV files will be created inside this directory, and their names can be controlled with the *fileNameFormat* setting.

Note that if the value specified here does not have a protocol, then it is assumed to be a file protocol. Relative URLs will be resolved against the current working directory. Also, for convenience, if the path begins with a tilde (~), that symbol will be expanded to the current user's home directory.

The options can be used in short form (`-k keyspace_name`) or long form (`--schema.keyspace keyspace_name`).

-c,--connector.name csv | json

The name of the connector to use.
Default: csv

Common to both CSV and JSON connectors

--connector.(csv|json).fileNameFormat *`string`*

The file name format to use when writing during unloading. This setting is ignored when reading and for non-file URLs. The file name must comply with the formatting

rules of `String.format()`, and must contain a `%d` format specifier that will be used to increment file name counters.
Default: output-%0,6d.csv (CSV); output-%0,6d.json (JSON)

--connector.(csv|json).fileNamePattern *`string`*

The glob pattern to use when searching for files to read. The syntax to use is the glob syntax, as described in `java.nio.file.FileSystem.getPathMatcher()`. This setting is ignored when writing and for non-file URLs. Only applicable when the *url* setting points to a directory on a known filesystem, ignored otherwise.
Default: **/*.csv (CSV); **/*.json (JSON)

-encoding,--connector.(csv|json).encoding *`string`*

The file encoding to use for all read or written files.
Default: UTF-8

-maxConcurrentFiles,--connector.(csv|json).maxConcurrentFiles *`string`*

The maximum number of files that can be written simultaneously. This setting is ignored when loading and when the output URL is anything other than a directory on a filesystem. The special syntax `NC` can be used to specify a number of threads that is a multiple of the number of available cores, e.g. if the number of cores is 8, then 0.5C = 0.5 * 8 = 4 threads. Used for unloading only.
Default: 0.25C

-maxRecords,--connector.(csv|json).maxRecords *`number`*

The maximum number of records to read from or write to each file. When reading, all records past this number will be discarded. When writing, a file will contain at most this number of records; if more records remain to be written, a new file will be created using the *fileNameFormat* setting. Note that when writing to anything other than a directory, this setting is ignored. For CSV, this setting takes into account the *header* setting: if a file begins with a header line, that line is not counted as a record. This feature is disabled by default (indicated by its `-1` value).
Default: -1

--connector.(csv|json).recursive (true | false)

Enable or disable scanning for files in the root's subdirectories. Only applicable when `url` is set to a directory on a known filesystem. Used for loading only.
Default: false

-skipRecords,--connector.(csv|json).skipRecords *`number`*

The number of records to skip from each input file before the parser can begin to execute. Note that if the file contains a header line (for CSV), that line is not counted as a valid record. Used for loading only.
Default: 0

-url,--connector.(csv|json).url *`string`*

The URL or path of the resource(s) to read from or wrote to. Possible options are `-` (representing stdin for reading and stdout for writing) and `file` (filepath). File URLs

can also be expressed as simple paths without the `file` prefix. A directory of files can also be specified.
Default: -

CSV connector options

-comment,--connector.csv.comment *`string`*

The character that represents a line comment when found in the beginning of a line of text. Only one character can be specified. Note that this setting applies to all files to be read or written.
Default: disabled by default and indicated with a `null` character value **"\u0000"**

-delim,--connector.csv.delimiter *`string`*

The character to use as field delimiter.
Default: , (a comma)

-escape,--connector.csv.escape *`string`*

The character used for escaping quotes inside an already quoted value. Only one character can be specified. Note that this setting applies to all files to be read or written.
Default: \

-header,--connector.csv.header (true | false)

Enable or disable whether the files to read or write begin with a header line. If enabled for loading, the first non-empty line in every file will assign field names for each record column, in lieu of schema.mapping *(page 25)*, `fieldA = col1, fieldB = col2, fieldC = col3`. If disabled for loading, records will not contain fields names, only field indexes, `0 = col1, 1 = col2, 2 = col3`. For unloading, if this setting is enabled, each file will begin with a header line, and if disabled, each file will not contain a header line.

Note: This option will apply to all files loaded or unloaded.

Default: true

--connector.csv.maxCharsPerColumn *`number`*

The maximum number of characters that a field can contain. This setting is used to size internal buffers and to avoid out-of-memory problems. If set to -1, internal buffers will be resized dynamically. While convenient, this can lead to memory problems. It could also hurt throughput, if some large fields require constant resizing; if this is the case, set this value to a fixed positive number that is big enough to contain all field values.
Default: 4096

--connector.csv.quote *`character`*

The character used for quoting fields when the field delimiter is part of the field value. Only one character can be specified. Note that this setting applies to all files to be read or written.

Default: \

JSON connector options

--connector.json.mode (SINGLE_DOCUMENT | MULTI_DOCUMENT)

The mode for loading and unloading JSON documents. Valid values are:

- MULTI_DOCUMENT: Each resource may contain an arbitrary number of successive JSON documents to be mapped to records. For example the format of each JSON document is a single document: `{doc1}`. The root directory for the JSON documents can be specified with `url` and the documents can be read recursively by setting `connector.json.recursive` to true.
- SINGLE_DOCUMENT: Each resource contains a root array whose elements are JSON documents to be mapped to records. For example, the format of the JSON document is an array with embedded JSON documents: `[ {doc1}, {doc2}, {doc3} ]`.

Default: MULTI_DOCUMENT

--connector.json.parserFeatures *map*

JSON parser features to enable. Valid values are all the enum constants defined in `com.fasterxml.jackson.core.JsonParser.Feature`. For example, a value of `{ ALLOW_COMMENTS : true, ALLOW_SINGLE_QUOTES : true }` will configure the parser to allow the use of comments and single-quoted strings in JSON data. Used for loading only.
Default: { }

--connector.json.generatorFeatures *map*

JSON generator features to enable. Valid values are all the enum constants defined in `com.fasterxml.jackson.core.JsonGenerator.Feature`. For example, a value of `{ ESCAPE_NON_ASCII : true, QUOTE_FIELD_NAMES : true }` will configure the generator to escape all characters beyond 7-bit ASCII and quote field names when writing JSON output. Used for unloading only.
Default: { }

--connector.json.serializationFeatures *map*

A map of JSON serialization features to set. Map keys should be enum constants defined in `com.fasterxml.jackson.databind.SerializationFeature`. Used for unloading only.
Default: { }

--connector.json.deserializationFeatures *map*

A map of JSON deserialization features to set. Map keys should be enum constants defined in `com.fasterxml.jackson.databind.DeserializationFeature`. The default value is the only way to guarantee that floating point numbers will not have their precision truncated when parsed, but can result in slightly slower parsing. Used for loading only.

Default: { USE_BIG_DECIMAL_FOR_FLOATS : true }

--connector.json.serializationStrategy *`string`*

The strategy for filtering out entries when formatting output. Valid values are enum constants defined in `com.fasterxml.jackson.annotation.JsonInclude.Include` Beware that the `CUSTOM` strategy cannot be honored). Used for unloading only.
Default: ALWAYS

--connector.json.prettyPrint (true | false)

Enable or disable pretty printing. When enabled, JSON records are written with indents. Used for unloading only.

Note: Can result in much bigger records.

Default: false

Count options

Specify count options for the dsbulk command. Count options specify how counting can be accomplished using dsbulk.

The options can be used in short form (`-k` *`keyspace_name`*) or long form (`--schema.keyspace` *`keyspace_name`*).

--stats.modes (global | ranges | hosts | partitions)

Kind(s) of statistics to compute. Only applicable for count, ignored otherwise. Valid values are:

- global: Count the total number of rows in the table.
- ranges: Count the total number of rows per token range in the table.
- hosts: Count the total number of rows per hosts in the table.
- partitions: Count the total number of rows in the N biggest partitions in the table. Choose how many partitions to track with `stats.numPartitions` option.

Default: global

--stats.numPartitions *`number`*

The number of distinct partitions for which to count rows. Only applicable for count, ignored otherwise.
10

Schema options

Specify schema options for the dsbulk command.

The options can be used in short form (`-k` *`keyspace_name`*) or long form (`--schema.keyspace` *`keyspace_name`*).

-k,--schema.keyspace *`string`*

Keyspace used for loading, unloading, or counting data. Keyspace names should not be quoted and are case-sensitive. `MyKeyspace` will match a keyspace named *MyKeyspace* but not *mykeyspace*. Required option if `schema.query` is not specified; otherwise, optional.
Default: unspecified

-t,--schema.table *`string`*

Table used for loading, unloading, or counting data. Table names should not be quoted and are case-sensitive. `MyTable` will match a table named *MyTable* but not *mytable*. Required option if `schema.query` is not specified; otherwise, optional.
Default: unspecified

-m,--schema.mapping *`string`*

The field-to-column mapping to use, that applies to both loading and unloading. If not specified, the loader will apply a strict one-to-one mapping between the source fields and the database table. If that is not what you want, then you must supply an explicit mapping. Mappings should be specified as a map of the following form:

- Indexed data sources: `0 = col1, 1 = col2, 2 = col3`, where `0`, `1`, `2`, are the zero-based indices of fields in the source data; and `col1`, `col2`, `col3` are bound variable names in the insert statement.
  - # A shortcut to map the first `n` fields is to simply specify the destination columns: `col1, col2, col3`.
- Mapped data sources: `fieldA = col1, fieldB = col2, fieldC = col3`, where `fieldA`, `fieldB`, `fieldC` are field names in the source data; and `col1`, `col2`, `col3` are bound variable names in the insert statement.

To specify that a field should be used for the query timestamp or ttl, use the specially named fake columns `__ttl` and `__timestamp`: `fieldA = __timestamp, fieldB = __ttl`. Timestamp fields can be parsed as CQL timestamp columns and must use the format specified in either codec.timestamp *(page 30)* or codec.unit *(page 32)* + codec.epoch *(page 32)*; the latter is an integer representing the number of units specified by codec.unit *(page 32)* since the specified epoch. TTL fields are parsed as integers representing duration in seconds and must use the format specified in codec.number *(page 29)*.

To specify that a column should be populated with the result of a function call, specify the function call as the input field (e.g. `now() = c4`). Note, this is relevant only for load operations.

In addition, for mapped data sources, it is also possible to specify that the mapping be partly auto-generated and partly explicitly specified. For example, if a source row has fields `c1`, `c2`, `c3`, and `c5`, and the table has columns `c1`, `c2`, `c3`, `c4`, one can map all like-named columns and specify that `c5` in the source maps to `c4` in the table as follows: `* = *, c5 = c4`.

One can specify that all like-named fields be mapped, except for `c2`: `* = -c2`. To skip `c2` and `c3`: `* = [-c2, -c3]`. Use quotes for any non-alphanumeric identifiers, fields or columns.

The exact type of mapping to use depends on the connector being used. Some connectors can only produce indexed records; others can only produce mapped ones, while others are capable of producing both indexed and mapped records at the same time. Refer to the connector's documentation to know which kinds of mapping it supports.
Default: unspecified

--schema.nullToUnset (true | false)

Specify whether to map `null` input values to "unset" in the database, i.e., don't modify a potentially pre-existing value of this field for this row. Valid for load scenarios, otherwise ignore. Note that setting to false creates tombstones to represent `null`.

Caution: This setting is applied after the *codec.nullStrings* setting, and may intercept `null`s produced by that setting.

Default: true

--schema.allowExtraFields (true | false)

Specify whether or not to accept records that contain extra fields that are not declared in the mapping. For example, if a record contains three fields A, B, and C, but the mapping only declares fields A and B, then if this option is true, C will be silently ignored and the record will be considered valid, and if false, the record will be rejected. Only applicable for loading, ignored otherwise.
Default: true

--schema.allowMissingFields (true | false)

Specify whether or not to accept records that are missing fields declared in the mapping. For example, if the mapping declares three fields A, B, and C, but a record contains only fields A and B, then if this option is true, C will be silently assigned null and the record will be considered valid, and if false, the record will be rejected. If the missing field is mapped to a primary key column, the record will always be rejected, since the database will reject the record. Only applicable for loading, ignored otherwise.
Default: false

-query,--schema.query *`string`*

The query to use. If not specified, then *schema.keyspace* and *schema.table* must be specified, and `dsbulk` will infer the appropriate statement based on the table's metadata, using all available columns. If `schema.keyspace` is provided, the query need not include the keyspace to qualify the table reference.

For loading, the statement can be any `INSERT`, `UPDATE` or `DELETE` statement. `INSERT` statements are preferred for most load operations, and bound variables should

correspond to mapped fields; for example, `INSERT INTO table1 (c1, c2, c3) VALUES (:fieldA, :fieldB, :fieldC)`. `UPDATE` statements are required if the target table is a counter table, and the columns are updated with incremental operations (`SET col1 = col1 + :fieldA`) where `:fieldA` is a column in the input data. A `DELETE` statement will remove existing data during the load operation.

For unloading, the statement can be any regular `SELECT` statement; it can optionally contain a token range restriction clause of the form: `token(...) > :start and token(...) <= :end`. If such a clause is present, the engine will generate as many statements as there are token ranges in the cluster, thus allowing parallelization of reads while at the same time targeting coordinators that are also replicas.

For counting, the statement must be a `SELECT` statement that contains only the table's partition key. The statement can optionally contain a token range restriction clause of the form: `token(...) > :start and token(...) <= :end`.

Statements can use both positional and named bound variables. Positional variables will be named after their corresponding column in the destination table. Named bound variables usually have names matching those of the columns in the destination table, but this is not a strict requirement; it is, however, required that their names match those of fields specified in the mapping.

See schema.mapping *(page 25)* setting for more information.
Default: unspecified

--schema.queryTimestamp *`string`*

The timestamp of inserted/updated cells during load; otherwise, the current time of the system running the tool is used. Not applicable to unloading nor counting. Express the value in ISO_ZONED_DATE_TIME format. DSE sets the query timestamps to the nearest microsecond and truncates sub-microseconds; any sub-microsecond information specified is lost. For more information, see the CQL Reference.
Default: unspecified

--schema.queryTtl *`number`*

The Time-To-Live (TTL) of inserted/updated cells during load (seconds); a value of -1 means there is no TTL. Not applicable to unloading. For more information, see the CQL Reference, Setting the time-to-live (TTL) for value, and Expiring data with time-to-live.
Default: -1

Batch options

Specify batch options for the dsbulk command. Batch options specify how statements are grouped before writing for loading. These options are not applicable for unloading.

The options can be used in short form (`-k` *`keyspace_name`*) or long form (`--schema.keyspace` *`keyspace_name`*).

--batch.bufferSize *number*

The buffer size to use for flushing batching statements. Do not set higher than `maxBatchSize` unless the loaded data is unsorted, when a higher value could improve performance. When set to a negative value the buffer size is implicitly set to `maxBatchSize`.
Default: -1

--batch.maxBatchSize *number*

The maximum batch size that depends on the size of the data inserted and the batch mode in use. Larger data requires a smaller value. For batch mode, `PARTITION_KEY` requires larger batch sizes, whereas `REPLICA_SET` requires smaller batch sizes, such as below 10.
Default: 32

--batch.mode *string*

The grouping mode. Valid values are:

- DISABLED: Disables statement batching.
- PARTITION_KEY: Groups together statements that share the same partition key. This is the default mode, and the preferred one.
- REPLICA_SET: Groups together statements that share the same replica set. This mode might yield better results for small clusters and lower replication factors, but tends to perform equally well or worse than `PARTITION_KEY` for larger clusters or high replication factors.

Default: PARTITION_KEY

Codec options

Specify codec options for the dsbulk command, which determine how record fields are parsed for loading or how row cells are formatted for unloading.

The options can be used in short form (`-k keyspace_name`) or long form (`--schema.keyspace keyspace_name`).

-locale,--codec.locale *string*

The locale to use for locale-sensitive conversions.
Default: en_US

-timeZone,--codec.timeZone *string*

The time zone to use for temporal conversions. When loading, the time zone will be used to obtain a timestamp from inputs that do not convey any explicit time zone information. When unloading, the time zone will be used to format all timestamps.
Default: UTC

-nullStrings,--codec.nullStrings *string*

Comma-separated list of strings that should be mapped to `null`. For loading, when a record field value exactly matches one of the specified strings, the value is replaced with `null` before writing to DSE. For unloading, this setting is only applicable for string-based connectors, such as the CSV connector: the first string specified will be used to change a row cell containing `null` to the specified string when written out. By default, no strings are mapped to `null`. If the target CQL type is textual (varchar or ascii), the original field value is untouched, and for other types, if the field value is an empty string, it is converted to `null` while loading, and `null` is converted to an empty string while unloading. This setting is applied before *schema.nullToUnset*, hence any `null` produced by a null-string can still be left unset if required.
Default: [ ] (no strings mapped to `null`)

--codec.booleanStrings *[true_value:false_value, ...]*

Specify how true and false representations can be used by dsbulk. Each representation is of the form `true_value:false_value`, case-insensitive. For loading, all representations are honored. For unloading, the first representation will be used and all others ignored.
Default: ["1:0","Y:N","T:F","YES:NO","TRUE:FALSE"]

--codec.booleanNumbers *[true_value, false_value]*

Set how true and false representations of numbers are interpreted. The representation is of the form `true_value,false_value`. The mapping is reciprocal, so that numbers are mapping to Boolean and vice versa. All numbers unspecified in this setting are rejected.
Default: [1, 0]

--codec.number *string*

The `DecimalFormat` pattern to use for conversion between `String` and CQL numeric types. See java.text.DecimalFormat for details about the pattern syntax to use. Most inputs are recognized: optional localized thousands separator, localized decimal separator, or optional exponent. Using locale *(page 28)* `en_US`, 1234, 1,234, 1234.5678, 1,234.5678 and 1,234.5678E2 are all valid. For unloading and formatting, rounding may occur and cause precision loss. See codec.formatNumbers *(page 29)* and codec.roundingStrategy *(page 30)*.
Default: #,###.##

--codec.formatNumbers (true | false)

Whether or not to use the `codec.number` pattern to format all numeric output. When set to `true`, the numeric pattern defined by `codec.number` will be applied. This allows for nicely-formatted output, but may result in rounding (see `codec.roundingStrategy`), or alteration of the original decimal's scale. When set to `false`, numbers will be stringified using the `toString` method, and will never result in rounding or scale alteration. Only applicable when unloading, and only if the connector in use requires stringification, because the connector, such as the CSV connector, does not handle raw numeric data; ignored otherwise.
Default: false

--codec.roundingStrategy *`string`*

The rounding strategy to use for conversions from CQL numeric types to String. Valid choices: any `java.math.RoundingMode` enum constant name, including: CEILING, FLOOR, UP, DOWN, HALF_UP, HALF_EVEN, HALF_DOWN, and UNNECESSARY. The precision used when rounding is inferred from the numeric pattern declared under `codec.number`. For example, the default `codec.number` (#,###.##) has a rounding precision of 2, and the number 123.456 would be rounded to 123.46 if the `codec.rounding Strategy` was set to `UP`. The default value will result in infinite precision, and ignore the `codec.number` setting. Only applicable when unloading, if `codec.formatNumbers` is true and if the connector in use requires stringification, because the connector, such as the CSV connector, does not handle raw numeric data; ignored otherwise.
Default: UNECESSARY

--codec.overflowStrategy *`string`*

This setting can mean one of three possibilities:

- The value is outside the range of the target CQL type. For example, trying to convert 128 to a CQL tinyint (max value of 127) results in overflow.
- The value is decimal, but the target CQL type is integral. For example, trying to convert 123.45 to a CQL int results in overflow.
- The value's precision is too large for the target CQL type. For example, trying to insert 0.1234567890123456789 into a CQL double results in overflow, because there are too many significant digits to fit in a 64-bit double.

Valid choices:

- REJECT: overflows are considered errors and the data is rejected. This is the default value.
- TRUNCATE: the data is truncated to fit in the target CQL type.

  Note: The truncation algorithm is similar to the narrowing primitive conversion defined in The Java Language Specification, Section 5.1.3, with the following exceptions: (1) If the value is too big or too small, it is rounded up or down to the maximum or minimum value allowed, rather than truncated at bit level. For example, 128 would be rounded down to 127 to fit in a byte, whereas Java would have truncated the exceeding bits and converted to -127 instead. (2) If the value is decimal, but the target CQL type is integral, it is first rounded to an integral using the defined rounding strategy, then narrowed to fit into the target type. This can result in precision loss and should be used with caution.

Only applicable for loading, when parsing numeric inputs; it does not apply for unloading, since formatting never results in overflow.
Default: REJECT

--codec.timestamp (*`formatter`* | *`string`*)

The temporal pattern to use for `String` to CQL timestamp conversion. Valid choices:

- A date-time pattern
- A pre-defined formatter such as `ISO_ZONED_DATE_TIME` or `ISO_INSTANT`, or any other public static field in java.time.format.DateTimeFormatter
- The special formatter `CQL_TIMESTAMP`, which is a special parser that accepts all valid CQL literal formats for the `timestamp` type.

Note: For more information on patterns and pre-defined formatters, see Patterns for Formatting and Parsing in Oracle Java documentation. For more information about CQL date, time and timestamp literals, see Date, time, and timestamp format.

When parsing, this format recognizes most CQL temporal literals:

Type	Values
Local dates	2012-01-01
Local times	12:34 12:34:56 12:34:56.123 12:34:56.123456 12:34:56.123456789
Local date-times	2012-01-01T12:34 2012-01-01T12:34:56 2012-01-01T12:34:56.123 2012-01-01T12:34:56.123456 2012-01-01T12:34:56.123456789
Zoned date-times	2012-01-01T12:34+01:00 2012-01-01T12:34:56+01:00 2012-01-01T12:34:56.123+01:00 2012-01-01T12:34:56.123456+01:00 2012-01-01T12:34:56.123456789+01:00 2012-01-01T12:34:56.123456789+01:00[Europe/Paris]

When the input is a local date, the timestamp is resolved at midnight using the time zone specified under `timeZone`. When the input is a local time, the timestamp is resolved using the time zone specified under `timeZone`, and the date is inferred from the instant specified under `epoch` (by default, January 1st 1970). When formatting, this format uses the `ISO_OFFSET_DATE_TIME` pattern, which is compliant with both CQL and ISO-8601.

Default: CQL_TIMESTAMP

--codec.date (*`formatter`* | *`string`*)

The temporal pattern to use for `String` to CQL date conversion. Valid choices:

- A date-time pattern
- A pre-defined formatter such as `ISO_LOCAL_DATE`

Note: For more information on patterns and pre-defined formatters, see Patterns for Formatting and Parsing in Oracle Java documentation. For more information about CQL date, time and timestamp literals, see Date, time, and timestamp format.

Default: ISO_LOCAL_DATE

--codec.time (*`formatter`* | *`string`*)

The temporal pattern to use for `String` to CQL time conversion. Valid choices:

- A date-time pattern, such as HH:mm:ss.
- A pre-defined formatter such as `ISO_LOCAL_TIME`

Note: For more information on patterns and pre-defined formatters, see Patterns for Formatting and Parsing in Oracle Java documentation. For more information about CQL date, time and timestamp literals, see Date, time, and timestamp format.

Default: ISO_LOCAL_TIME

--codec.unit

This setting applies only to CQL timestamp columns, and `USING TIMESTAMP` clauses in queries. If the input is a string containing only digits that cannot be parsed using the codec.timestamp *(page 30)* format, the specified time unit is applied to the parsed value. All `TimeUnit` enum constants are valid choices.
Default: MILLISECONDS

--codec.epoch

This setting applies only to CQL timestamp columns, and `USING TIMESTAMP` clauses in queries. If the input is a string containing only digits that cannot be parsed using the codec.timestamp *(page 30)* format, the specified epoch determines the relative point in time used with the parsed value. The value must be a valid timestamp, defined by options `codec.timestamp` and codec.timeZone *(page 28)* (if the value does not include a time zone).
Default: 1970-01-01T00:00:00Z

--codec.uuidStrategy (RANDOM | FIXED | MIN | MAX)

Strategy to use when generating time-based (version 1) UUIDs from timestamps. Clock sequence and node ID parts of generated UUIDs are determined on a best-effort basis and are not fully compliant with RFC 4122. Valid values are:

- RANDOM: Generates UUIDs using a random number in lieu of the local clock sequence and node ID. This strategy will ensure that the generated UUIDs are unique, even if the original timestamps are not guaranteed to be unique.
- FIXED: Preferred strategy if original timestamps are guaranteed unique, since it is faster. Generates UUIDs using a fixed local clock sequence and node ID.
- MIN: Generates the smallest possible type 1 UUID for a given timestamp.

Warning: This strategy doesn't guarantee uniquely generated UUIDs and should be used with caution.

- MAX: Generates the biggest possible type 1 UUID for a given timestamp.

  Warning: This strategy doesn't guarantee uniquely generated UUIDs and should be used with caution.

Default: RANDOM

Driver options

Specify driver options https://docs.datastax.com/en/developer/driver-matrix/doc/common/driverMatrix.html for the dsbulk command.

The options can be used in short form (`-k keyspace_name`) or long form (`--schema.keyspace keyspace_name`).

General driver options

-h,--driver.hosts *`host_name(s)`*

The contact points to use for the initial connection to the cluster. This must be a comma-separated list of hosts, each specified by a host-name or IP address. If the host is a DNS name that resolves to multiple A-records, all the corresponding addresses will be used. Do not use `localhost` as a host-name (since it resolves to both IPv4 and IPv6 addresses on some platforms). The port for all hosts must be specified with `driver.port`.
Default: 127.0.0.1

-port,--driver.port *`port_number`*

The port to connect to at initial contact points. Note that all nodes in a cluster must accept connections on the same port number.
Default: 9042

--driver.addressTranslator *`string`*

The simple or fully-qualified class name of the address translator to use. This is only needed if the nodes are not directly reachable from the machine on which dsbulk is running (for example, the dsbulk machine is in a different network region and needs to use a public IP, or it connects through a proxy).
Default: IdentityTranslator

--driver.timestampGenerator (AtomicMonotonicTimestampGenerator | ThreadLocalTimestampGenerator | ServerSideTimestampGenerator)

The simple or fully-qualified class name of the timestamp generator to use. Built-in options are:

- AtomicMonotonicTimestampGenerator: timestamps are guaranteed to be unique across all client threads.

- ThreadLocalTimestampGenerator: timestamps are guaranteed to be unique within each thread only.
- ServerSideTimestampGenerator: do not generate timestamps, let the server assign them.

Default: AtomicMonotonicTimestampGenerator

Driver policies

Driver policy options pertain to DSE Java Driver load balancing policy settings. See Load Balancing for details.

-lbp,--driver.policy.lbp.name (dse | dcAwareRoundRobin | roundRobin | whiteList | tokenAware)

The name of the load balancing policy. Supported policies include: `dse`, `dcAwareRoundRobin`, `roundRobin`, `whiteList`, `tokenAware`. Available options for the policies are listed below as appropriate. For more information, refer to the driver documentation for the policy. If not specified, defaults to the driver's default load balancing policy, which is currently the `DseLoadBalancingPolicy` wrapping a `TokenAwarePolicy`, wrapping a `DcAwareRoundRobinPolicy`.

Note: It is critical for a token-aware policy to be used in the chain in order to benefit from batching by partition key.

Default: unspecified

--driver.policy.lbp.dcAwareRoundRobin.allowRemoteDCsForLocalConsistencyLevel (true | false)

Enable or disable whether to allow remote datacenters to count for local consistency level in round robin awareness.
Default: false

--driver.policy.lbp.dcAwareRoundRobin.localDc *`string`*

The datacenter name (commonly dc1, dc2, etc.) local to the machine on which dsbulk is running, so that requests are sent to nodes in the local datacenter whenever possible.
Default: unspecified

--driver.policy.lbp.dcAwareRoundRobin.usedHostsPerRemoteDc *`number`*

The number of hosts per remote datacenter that the round robin policy should consider.
Default: 0

--driver.policy.lbp.dse.childPolicy (dse | dcAwareRoundRobin | roundRobin | whiteList | tokenAware)

The child policy that the specified `dse` policy wraps.
Default: roundRobin

--driver.policy.lbp.tokenAware.childPolicy (dse | dcAwareRoundRobin | roundRobin | whiteList | tokenAware)

The child policy that the specified `tokenAware` policy wraps.
Default: roundRobin

--driver.policy.lbp.tokenAware.shuffleReplicas (true | false)

Specify whether to shuffle the list of replicas that can process a request. For loading, shuffling can improve performance by distributing writes across nodes.
Default: true

--driver.policy.lbp.whiteList.childPolicy (dse | dcAwareRoundRobin | roundRobin | whiteList | tokenAware)

The child policy that the specified `whiteList` policy wraps.
Default: roundRobin

--driver.policy.lbp.whiteList.hosts *`string`*

List of hosts to white list. This must be a comma-separated list of hosts, each specified by a host-name or ip address. If the host is a DNS name that resolves to multiple A-records, all the corresponding addresses will be used. Do not use `localhost` as a host-name (since it resolves to both IPv4 and IPv6 addresses on some platforms).
Default: unspecified

-maxRetries,--driver.policy.maxRetries *`number`*

Maximum number of retries for a timed-out request.
Default: 10

Driver pooling

--driver.pooling.heartbeat *`string`*

The heartbeat interval. If a connection stays idle for that duration (no reads), the driver sends a dummy message on it to make sure it's still alive. If not, the connection is trashed and replaced.
Default: 30 seconds

--driver.pooling.local.connections *`number`*

The number of connections in the pool for nodes at "local" distance.
Default: 4

--driver.pooling.local.requests *`number`*

The maximum number of requests (1 to 32768) that can be executed concurrently on a connection.
Default: 32768

--driver.pooling.remote.connections *`number`*

The number of connections in the pool for remote nodes.

Default: 1

--driver.pooling.remote.requests *number*

The maximum number of requests (1 to 32768) that can be executed concurrently on a connection.
Default: 1024

Driver compression protocol

--driver.protocol.compression (NONE | LZ4 | SNAPPY)

Specify the compression algorithm to use.
Default: NONE

Driver query

-cl,--driver.query.consistency (ANY | LOCAL_ONE | ONE | TWO | THREE | LOCAL_QUORUM | QUORUM | EACH_QUORUM | ALL)

The consistency level to use for both loading and unloading. Note that stronger consistency levels usually result in reduced throughput In addition, any level higher than `ONE` will automatically disable continuous paging, which can dramatically reduce read throughput.
Default: LOCAL_ONE

--driver.query.fetchSize *number*

The page size, or how many rows will be retrieved simultaneously in a single network round trip. This setting will limit the number of results loaded into memory simultaneously during unloading. Not applicable for loading.

Default: 5000

--driver.query.idempotence (true | false)

The default idempotence of statements generated by the loader.
Default: true

--driver.query.serialConsistency (SERIAL | LOCAL_SERIAL)

The serial consistency level to use for writes. Only applicable if the data is inserted using lightweight transactions, ignored otherwise.
Default: LOCAL_SERIAL

Driver socket

--driver.socket.readTimeout *string*

The time the driver waits for a request to complete. This is a global limit on the duration of a `session.execute()` call, including any internal retries the driver might do.

Default: 60 seconds

Security options

Specify authorization and SSL encryption options for the dsbulk command. For additional information on SSL, see the Oracle Java Guide on SSL.

The options can be used in short form (`-k keyspace_name`) or long form (`--schema.keyspace keyspace_name`).

Authorization options

--driver.auth.provider (None | PlainTextAuthProvider | DsePlainTextAuthProvider | DSEGSSAPIAuthProvider)

The name of the AuthProvider to use. Valid choices are:

- None: no authentication.
- PlainTextAuthProvider: Uses `com.datastax.driver.core.PlainTextAuthProvider` for authentication. Supports SASL authentication using the `PLAIN` mechanism (plain text authentication).
- DsePlainTextAuthProvider: Uses `com.datastax.driver.dse.auth.DsePlainTextAuthProvider` for authentication. Supports SASL authentication to DSE clusters using the `PLAIN` mechanism (plain text authentication), and also supports optional proxy authentication; should be preferred to `PlainTextAuthProvider` when connecting to secured DSE clusters.
- DseGSSAPIAuthProvider: Uses `com.datastax.driver.dse.auth.DseGSSAPIAuthProvider` for authentication. Supports SASL authentication to DSE clusters using the `GSSAPI` mechanism (Kerberos authentication), and also supports optional proxy authentication.

  Note: When using this provider you may have to set the `java.security.krb5.conf` system property to point to your `krb5.conf` file (e.g. set the `DSBULK_JAVA_OPTS` environment variable to `-Djava.security.krb5.conf=/home/user/krb5.conf`). See the Oracle Java Kerberos documentation for more details.

Default: None

-u,--driver.auth.username *`string`*

The username to use. Providers that accept this setting:

- `PlainTextAuthProvider`

- `DsePlainTextAuthProvider`

Default: unspecified

-p,--driver.auth.password *`string`*

The password to use. Providers that accept this setting:

- `PlainTextAuthProvider`
- `DsePlainTextAuthProvider`

Default: unspecified

--driver.auth.authorizationId *`string`*

An authorization ID allows the currently authenticated user to act as a different user (proxy authentication). Providers that accept this setting:

- `DsePlainTextAuthProvider`
- `DseGSSAPIAuthProvider`

Default: unspecified

--driver.auth.keyTab *`string`*

The path of the Kerberos keytab file to use for authentication. If left unspecified, authentication uses a ticket cache. Providers that accept this setting:

- `DseGSSAPIAuthProvider`

Default: unspecified

--driver.auth.principal *`email`*

The Kerberos principal to use. For example, `user@datastax.com`. If left unspecified, the principal is chosen from the first key in the ticket cache or keytab. Providers that accept this setting:

- `DseGSSAPIAuthProvider`

Default: unspecified

--driver.auth.saslService*`string`*

The SASL service name to use. This value should match the username of the Kerberos service principal used by the DSE server. This information is specified in the `dse.yaml` file by the *service_principal* option under the *kerberos_options* section, and may vary from one DSE installation to another – especially if you installed DSE with an automated package installer. Providers that accept this setting:

- `DseGSSAPIAuthProvider`

Default: dse

SSL encryption options

--driver.ssl.cipherSuites *`list`*

The cipher suites to enable. For example:

```
cipherSuites = ["TLS_RSA_WITH_AES_128_CBC_SHA",
 "TLS_RSA_WITH_AES_256_CBC_SHA"]
```

This property is optional. If it is not present, the driver won't explicitly enable cipher suites, which according to the JDK documentation results in "a minimum quality of service".
Default: [ ] (empty list)

--driver.ssl.keystore.algorithm (SunX509 | NewSunX509)

The algorithm to use for the SSL keystore.
Default: SunX509

--driver.ssl.keystore.password *string*

The keystore password.
Default: unspecified

--driver.ssl.keystore.path *string*

The path of the keystore file. This setting is optional. If left unspecified, no client authentication will be used.
Default: unspecified

--driver.ssl.openssl.keyCertChain *string*

The path of the certificate chain file. This setting is optional. If left unspecified, no client authentication will be used.
Default: unspecified

--driver.ssl.openssl.privateKey *string*

The path of the private key file.
Default: unspecified

--driver.ssl.provider (None | JDK | OpenSSL)

The SSL provider to use. *JDK* uses the JDK SSLContext and *OpenSSL* uses Netty native support for OpenSSL. Using OpenSSL provides better performance and generates less garbage. This is the recommended provider when using SSL.
Default: None

--driver.ssl.truststore.algorithm (PKIX | SunX509)

The algorithm to use for the SSL truststore.
Default: SunX509

--driver.ssl.truststore.password *string*

The truststore password.
Default: unspecified

--driver.ssl.truststore.path *string*

The path of the truststore file. This setting is optional. If left unspecified, server certificates will not be validated.
Default: unspecified

Engine options

Specify engine options for the dsbulk command.

The options can be used in short form (`-k keyspace_name`) or long form (`--schema.keyspace keyspace_name`).

-dryRun,--engine.dryRun (true | false)

Enable or disable dry-run mode, a test mode that runs the command but does not load data. Not applicable for unloading.
Default: false

--engine.executionId *`string`*

A unique identifier to attribute to each execution. When unspecified or empty, the engine will automatically generate identifiers of the following form: *`workflow_timestamp`*, where :

- *`workflow`* stands for the workflow type (`LOAD`, `UNLOAD`, etc.);
- *`timestamp`* is the current timestamp formatted as `uuuuMMdd-HHmmss-SSSSSS` (see Patterns for Formatting and Parsing in Oracle Java documentation) in UTC, with microsecond precision if available, and millisecond precision otherwise.

When this identifier is user-supplied, it is important to guarantee its uniqueness; failing to do so may result in execution failures. It is also possible to provide templates here. Any format compliant with the formatting rules of `String.format()` is accepted, and can contain the following parameters:

- `%1$s` : the workflow type (`LOAD`, `UNLOAD`, etc.);
- `%2$t` : the current time (with microsecond precision if available, and millisecond precision otherwise);
- `%3$s` : the JVM process PID (this parameter might not be available on some operating systems; if its value cannot be determined, a random integer will be inserted instead).

Default: unspecified

Executor options

Specify executor options for the dsbulk command.

The options can be used in short form (`-k keyspace_name`) or long form (`--schema.keyspace keyspace_name`).

--executor.maxPerSecond *`number`*

The maximum number of concurrent operations per second. This acts as a safeguard to prevent more requests than the cluster can handle. Batch statements are counted by the number of statements included. Reduce this setting when the latencies get too high and a remote cluster cannot keep up with throughput, as `dsbulk` requests will eventually time out. Setting this option to any negative value will disable it.
Default: -1

--executor.continuousPaging.enabled (true | false)

Enable or disable continuous paging. If the target cluster does not support continuous paging, or if driver.query.consistency *(page 36)* is not `ONE` or `LOCAL_ONE`, traditional paging will be used regardless of this setting. Used for unloading only.
Default: true

--executor.continuousPaging.maxPages *`number`*

The maximum number of pages to retrieve. Setting this value to zero retrieves all pages available.
Default: 0

--executor.continuousPaging.maxPagesPerSecond *`number`*

The maximum number of pages per second. Setting this value to zero indicates no limit.
Default: 0

--executor.continuousPaging.pageSize *`number`*

The size of the page. The unit to use is determined by the `pageUnit` setting.
Default: 5000

--executor.continuousPaging.pageUnit (ROWS | BYTES)

The unit to use for the `pageSize` setting.
Default: ROWS

--executor.maxInFlight *`number`*

The maximum number of "in-flight" requests, or maximum number of concurrent requests waiting for a response from the server. This acts as a safeguard to prevent more requests than the cluster can handle. Batch statements count as one request. Reduce this value when the throughput for reads and writes cannot match the throughput of mappers; this is usually a sign that the workflow engine is not well calibrated and will eventually run out of memory. Setting this option to any negative value will disable it.

Default: 1024

Logging options

Specify logging and error options for the dsbulk command. Log messages are only logged to the main log file, `operation.log`, and standard error, and nothing is printed to `stdout`.

The options can be used in short form (`-k keyspace_name`) or long form (`--schema.keyspace keyspace_name`).

-ansiMode,--log.ansiMode (normal | force | disable)

Whether or not to use ANSI colors and other escape sequences in log messages printed to the console. By default, `dsbulk` uses colored output (`normal`) when the terminal is: (1) compatible with ANSI escape sequences; all common terminals on *nix and BSD systems, including MacOS, are ANSI-compatible, and some popular terminals for Windows (Mintty, MinGW) or (2) a standard Windows DOS command prompt (ANSI sequences are translated on the fly). The `force` value will cause `dsbulk` to use ANSI colors even for non ANSI-compatible terminals detected. There should be no reason to disable ANSI escape sequences, but if, for some reason, colored messages are not desired or not printed correctly, this option allows disabling ANSI support altogether. For Windows: ANSI support works best with the (Microsoft Visual C++ 2008 SP1 Redistributable Package) installed.
Default: normal

-maxErrors,--log.maxErrors (*`number`* | *`"N%"`*)

The maximum number of errors to tolerate before aborting the entire operation. Set to either a number or a string of the form `N%` where `N` is a decimal number between 0 and 100. Setting this value to `-1` disables this feature (not recommended).
Default: 10

-logDir,--log.directory *`path_to_directory`*

The writable directory where all log files will be stored; if the directory specified does not exist, it will be created. URLs are not acceptable (not even `file:/` URLs). Log files for a specific run, or execution, will be located in a sub-directory under the specified directory. Each execution generates a sub-directory identified by an "execution ID". See engine.executionId *(page 40)* for more information about execution IDs. Relative paths will be resolved against the current working directory. Also, for convenience, if the path begins with a tilde (~), that symbol will be expanded to the current user's home directory.
Default: ./logs

--log.stmt.level (ABRIDGED | NORMAL | EXTENDED |

The desired log level for printing to log files. Valid values are:

- ABRIDGED: Print only basic information in summarized form.

- NORMAL: Print basic information in summarized form, and the statement's query string, if available. For batch statements, this verbosity level also prints information about the batch's inner statements.
- EXTENDED: Print full information, including the statement's query string, if available, and the statement's bound values, if available. For batch statements, this verbosity level also prints all information available about the batch's inner statements.

Default: EXTENDED

--log.stmt.maxBoundValueLength *number*

The maximum length for a bound value. Bound values longer than this value will be truncated.

Important: Setting this value to `-1` disables this feature (not recommended).

Default: 50

--log.stmt.maxBoundValues *number*

The maximum number of bound values to print. If the statement has more bound values than this limit, the exceeding values will not be printed.

Important: Setting this value to `-1` disables this feature (not recommended).

Default: 50

--log.stmt.maxInnerStatements *number*

The maximum number of inner statements to print for a batch statement. Only applicable for batch statements, ignored otherwise. If the batch statement has more children than this value, the exceeding child statements will not be printed.

Important: Setting this value to `-1` disables this feature (not recommended).

Default: 10

--log.stmt.maxQueryStringLength *number*

The maximum length for a query string. Query strings longer than this value will be truncated.

Important: Setting this value to `-1` disables this feature (not recommended).

Default: 500

-verbosity, --log.verbosity (0 | 1 | 2)

Desired level of verbosity. Valid values are:

- 0 (quiet): Only log WARN and ERROR messages.
- 1 (normal): Log INFO, WARN and ERROR messages.
- 2 (verbose) Log DEBUG, INFO, WARN and ERROR messages.

Default: 1

Monitoring options

Monitor throughput is measured as operations/sec, an operation being a single write or a single read. But this unit of measurement can vary greatly, depending on the size of the row being written or read. A different measure of monitor throughput is mb/sec, to combat the irregularity of measuring by operations/sec. In a load work flow, a typical report shows:

```
2018-03-14 13:15:48 INFO Memory usage: used: 507 MB, free: 691 MB,
 allocated: 1,199 MB, available: 3,641 MB,
    total gc count: 20, total gc time: 346 ms
2018-03-14 13:15:48 INFO Records: total: 210,755, successful: 210,755,
 failed: 0, mean: 20,893 records/second
2018-03-14 13:15:48 INFO Batches: total: 6,602, size: 31.90 mean, 10 min,
 32 max
2018-03-14 13:15:48 INFO Writes: total: 210,669, successful: 210,669,
 failed: 0, in-flight: 0
2018-03-14 13:15:48 INFO Throughput: 20,877 writes/second, 1.11 mb/second
2018-03-14 13:15:48 INFO Latencies: mean 6.29, 75p 2.87, 99p 89.13, 999p
 125.83 milliseconds
```

Specify monitoring options for the dsbulk command.

The options can be used in short form (`-k` *`keyspace_name`*) or long form (`--schema.keyspace` *`keyspace_name`*).

-reportRate,--monitoring.reportRate *`string`*

The report interval for the console reporter. The console reporter will print useful metrics about the ongoing operation at this rate. Durations lesser than one second will be rounded up to 1 second.
Default: 5 seconds

--monitoring.csv (true | false)

Enable or disable CSV reporting. If enabled, CSV files containing metrics will be generated in the designated log directory.
Default: false

--monitoring.durationUnit *`string`*

The time unit used when printing latency durations. Valid values: all `TimeUnit` enum constants.
Default: MILLISECONDS

--monitoring.expectedReads *`number`*

The expected total number of reads. Optional, but if set, the console reporter will also print the overall achievement percentage. Setting this value to `-1` disables this feature.
Default: -1

--monitoring.expectedWrites *`number`*

The expected total number of writes. Optional, but if set, the console reporter will also print the overall achievement percentage. Setting this value to `-1` disables this feature.
Default: -1

-jmx,--monitoring.jmx (true | false)

Enable or disable JMX reporting. Note that to enable remote JMX reporting, several properties must also be set in the JVM during launch. This is accomplished via the `DSBULK_JAVA_OPTS` environment variable.
Default: true

--monitoring.rateUnit *`string`*

The time unit used when printing throughput rates. Valid values: all `TimeUnit` enum constants.
Default: SECONDS