Package ‘sergeant’

July 18, 2017

Title Tools to Transform and Query Data with 'Apache' 'Drill'

Version 0.5.2

Description 'Apache Drill' is a low-latency distributed query engine designed to enable data exploration and 'analytics' on both relational and non-relational 'datastores', scaling to petabytes of data. Methods are provided that enable working with 'Apache' 'Drill' instances via the 'REST' 'API', 'JDBC' interface (optional), 'DBI' 'methods' and using 'dplyr'/dbplyr' idioms.

Depends R (>= 3.1.2), DBI (>= 0.7), dplyr (>= 0.7.0), dbplyr (>= 1.1.0)

URL https://github.com/hrbrmstr/sergeant

BugReports https://github.com/hrbrmstr/sergeant/issues

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Imports httr (>= 1.2.1), jsonlite (>= 1.5.0), htmltools (>= 0.3.6), readr (>= 1.1.1), purrr (>= 0.2.2), scales (>= 0.4.1), utils, methods

Suggests RJDBC (>= 0.2-5), rJava (>= 0.9-8), testthat (>= 1.0.2), covr (>= 3.0.0)

RoxygenNote 6.0.1

NeedsCompilation no

Author Bob Rudis [aut, cre], Edward Visel [ctb]

Maintainer Bob Rudis <bob@rud.is>

Repository CRAN

Date/Publication 2017-07-17 22:36:26 UTC
### R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbDataType,DrillConnection-method</td>
<td>2</td>
</tr>
<tr>
<td>dbUnloadDriver,DrillDriver-method</td>
<td>3</td>
</tr>
<tr>
<td>Drill</td>
<td>3</td>
</tr>
<tr>
<td>drill_active</td>
<td>4</td>
</tr>
<tr>
<td>drill_cancel</td>
<td>4</td>
</tr>
<tr>
<td>drill_connection</td>
<td>5</td>
</tr>
<tr>
<td>drill_custom_functions</td>
<td>5</td>
</tr>
<tr>
<td>drill_jdbc</td>
<td>7</td>
</tr>
<tr>
<td>drill_metrics</td>
<td>8</td>
</tr>
<tr>
<td>drill_options</td>
<td>9</td>
</tr>
<tr>
<td>drill_profile</td>
<td>9</td>
</tr>
<tr>
<td>drill_profiles</td>
<td>10</td>
</tr>
<tr>
<td>drill_query</td>
<td>10</td>
</tr>
<tr>
<td>drill_set</td>
<td>11</td>
</tr>
<tr>
<td>drill_settings_reset</td>
<td>12</td>
</tr>
<tr>
<td>drill_show_files</td>
<td>12</td>
</tr>
<tr>
<td>drill_show_schemas</td>
<td>13</td>
</tr>
<tr>
<td>drill_stats</td>
<td>13</td>
</tr>
<tr>
<td>drill_status</td>
<td>14</td>
</tr>
<tr>
<td>drill_storage</td>
<td>14</td>
</tr>
<tr>
<td>drill_system_reset</td>
<td>15</td>
</tr>
<tr>
<td>drill_threads</td>
<td>16</td>
</tr>
<tr>
<td>drill_uplift</td>
<td>16</td>
</tr>
<tr>
<td>drill_use</td>
<td>17</td>
</tr>
<tr>
<td>drill_version</td>
<td>17</td>
</tr>
<tr>
<td>sergeant</td>
<td>18</td>
</tr>
<tr>
<td>sergeant-exports</td>
<td>18</td>
</tr>
<tr>
<td>src_drill</td>
<td>19</td>
</tr>
</tbody>
</table>

### Index

21

---

**dbDataType,DrillConnection-method**

Drill dbDataType

### Description

Drill dbDataType

### Usage

```r
## S4 method for signature 'DrillConnection'
dbDataType(dbObj, obj, ...)
```
**Args**

- `dbobj`: A `DrillDriver` object
- `obj`: Any R object
- `...`: Extra optional parameters

---

**Description**

Unload driver

**Usage**

```r
## S4 method for signature 'DrillDriver'
dbUnloadDriver(drv, ...)
```

**Args**

- `drv`: driver
- `...`: Extra optional parameters

---

**Drill**

**Description**

Drill

Connect to Drill

**Usage**

```r
Drill()
```

```
## S4 method for signature 'DrillDriver'
dbConnect(drv, host = "localhost", port = 8047L,
          ssl = FALSE, ...)
```

**Args**

- `drv`: An object created by `Drill()`
- `host`: host
- `port`: port
- `ssl`: use ssl?
- `...`: Extra optional parameters
drill_active  Test whether Drill HTTP REST API server is up

Description
This is a very simple test (performs HEAD / request on the Drill server/cluster)

Usage
```r
  drill_active(drill_con)
```

Arguments
- `drill_con` drill server connection object setup by `drill_connection()`

Examples
```r
## Not run:
  drill_connection() %>% drill_active()
## End(Not run)
```

---

drill_cancel  Cancel the query that has the given queryid

Description
Cancel the query that has the given queryid

Usage
```r
  drill_cancel(drill_con, query_id)
```

Arguments
- `drill_con` drill server connection object setup by `drill_connection()`
- `query_id` the UUID of the query in standard UUID format that Drill assigns to each query.

References
- Drill documentation
**drill_connection**  
Setup a Drill connection

**Description**
Setup a Drill connection

**Usage**
```r
drill_connection(host =Sys.getenv("DRILL_HOST", "localhost"),  
                 port =Sys.getenv("DRILL_PORT", 8047), ssl = FALSE,  
                 user =Sys.getenv("DRILL_USER", ""),  
                 password =Sys.getenv("DRILL_PASSWORD", ""))
```

**Arguments**
- host: Drill host (will pick up the value from DRILL_HOST env var)
- port: Drill port (will pick up the value from DRILL_PORT env var)
- ssl: use ssl?
- user, password: NOT IMPLEMENTED YET credentials for username/password auth. (will pick up the values from DRILL_USER/DRILL_PASSWORD env vars)

**Examples**
```r
dc <- drill_connection()
```

**drill_custom_functions**  
Drill expressions / custom functions dplyr translations

**Description**
One benefit of dplyr is that it provide a nice DSL over database ops but that means there needs to be knowlege of functions supported by the host database and then a translation layer so they can be used in R.

**Details**
Similarly, there are functions like grepl() in R that don’t directly exist in databases. Yet, one can create a translation for grepl() that maps to a Drill custom function so you don’t have to think differently or rewrite your pipes when switching from core tidyverse ops and database ops.
Many functions translate on their own, but it’s handy to provide explicit ones, especially when you want to use parameters in a different order.
If you want a particular custom function mapped, file a PR or issue request in the link found in the DESCRIPTION file.
• as.character(x) : CAST( x AS CHARACTER )
• as.date(x) : CAST( x AS DATE )
• as.logical(x) : CAST( x AS BOOLEAN)
• as.numeric(x) : CAST( x AS DOUBLE )
• as.posixct(x) : CAST( x AS TIMESTAMP )
• binary_string(x) : BINARY_STRING( x )
• cbrt(x) : CBRT( x )
• char_to_timestamp(x, y) : TO_TIMESTAMP( x, y )
• grepl(y, x) : CONTAINS( x, y )
• contains(x, y) : CONTAINS( x, y )
• convert_to(x, y) : CONVERT_TO( x, y )
• convert_from(x, y) : CONVERT_FROM( x, y )
• degrees(x) : DEGREES( x )
• lshift(x, y) : DEGREES( x, y )
• negative(x) : NEGATIVE( x )
• pow(x, y) : MOD( x, y )
• sql_prefix(x, y) : POW( x, y )
• string_binary(x) : STRING_BINARY( x )
• radians(x) : RADIANS( x )
• rshift(x) : RSHIFT( x )
• to_char(x, y) : TO_CHAR x, y )
• to_date(x, y) : TO_DATE( x, y )
• to_number(x, y) : TO_NUMBER( x, y )
• trunc(x) : TRUNC( x )
• double_to_timestamp(x) = TO_TIMESTAMP( x )
• char_length(x) = CHAR_LENGTH( x )
• flatten(x) = FLATTEN( x )
• kvgen(x) = KVGEN( x )
• repeated_count(x) = REPEATED_COUNT( x )
• repeated_contains(x) = REPEATED_CONTAINS( x )
• ilike(x, y) = I LIKE( x, y )
• init_cap(x) = INIT_CAP( x )
• length(x) = LENGTH( x )
• lower(x) = LOWER( x )
• tolower(x) = LOWER( x )
• ltrim(x, y) = LTRIM( x, y )
• nullif(x, y) = NULLIF( x, y )
You can get a compact list of these with:
sql_translate_env(src_drill())$con
as well.

---

drill_jdbc

Connect to Drill using JDBC

Description

The DRILL JDBC driver fully-qualified path must be placed in the DRILL_JDBC_JAR environment variable. This is best done via ~/.Renviron for interactive work. e.g. DRILL_JDBC_JAR=/usr/local/drill/jars/jdbc-driver

Usage

drill_jdbc(nodes = "localhost:2181", cluster_id = NULL, schema = NULL, use_zk = TRUE)

Arguments

nodes character vector of nodes. If more than one node, you can either have a single string with the comma-separated node:port pairs pre-made or pass in a character vector with multiple node:port strings and the function will make a comma-separated node string for you.

cluster_id the cluster id from drill-override.conf

schema an optional schema name to append to the JDBC connection string

use_zk are you connecting to a ZooKeeper instance (default: TRUE) or connecting to an individual DrillBit.
drill_metrics

Value

a JDBC connection object

References


Examples

```r
## Not run:
con <- drill_jdbc("localhost:2181", "main")
drill_query(con, "SELECT * FROM cp.`employee.json`")

# you can also use the connection with RJDBC calls:
dbGetQuery(con, "SELECT * FROM cp.`employee.json`")

# for local/embedded mode with default configuration info
con <- drill_jdbc("localhost:31010", use_zk=FALSE)

## End(Not run)
```

drill_metrics  Get the current memory metrics

Description

Get the current memory metrics

Usage

`drill_metrics(drill_con)`

Arguments

`drill_con`  drill server connection object setup by `drill_connection()`

Examples

```r
## Not run:
drill_connection() %>% drill_metrics()

## End(Not run)
```
**drill_options**

List the name, default, and data type of the system and session options

**Description**

List the name, default, and data type of the system and session options

**Usage**

`drill_options(drill_con, pattern = NULL)`

**Arguments**

- `drill_con`: drill server connection object setup by `drill_connection()`
- `pattern`: pattern to filter results by

**References**

Drill documentation

**Examples**

```
## Not run:
drill_connection() %>% drill_options()

## End(Not run)
```

**drill_profile**

Get the profile of the query that has the given queryid

**Description**

Get the profile of the query that has the given queryid

**Usage**

`drill_profile(drill_con, query_id)`

**Arguments**

- `drill_con`: drill server connection object setup by `drill_connection()`
- `query_id`: UUID of the query in standard UUID format that Drill assigns to each query

**References**

Drill documentation
drill_profiles  
*Get the profiles of running and completed queries*

**Description**

Get the profiles of running and completed queries

**Usage**

```r
drill_profiles(drill_con)
```

**Arguments**

- `drill_con`: drill server connection object setup by `drill_connection()`

**References**

Drill documentation

**Examples**

```r
## Not run:
drill_connection() %>% drill_profiles()

## End(Not run)
```

---

**drill_query**  
*Submit a query and return results*

**Description**

This function can handle REST API connections or JDBC connections. There is a benefit to calling this function for JDBC connections vs a straight call to `dbGetQuery()` in that the function result is a ‘tbl_df’ vs a plain `data.frame` so you get better default printing (which can be helpful if you accidentally execute a query and the result set is huge).

**Usage**

```r
drill_query(drill_con, query, uplift = TRUE, .progress = interactive())
```

**Arguments**

- `drill_con`: drill server connection object setup by `drill_connection()` or `drill_jdbc()`
- `query`: query to run
- `uplift`: automatically run `drill_uplift()` on the result? (default: TRUE, ignored if `drill_con` is a JDBCConnection created by `drill_jdbc()`)  
  - If TRUE (default if in an interactive session) then ask `httr::POST` to display a progress bar
- `progress`: if TRUE (default if in an interactive session) then ask `httr::POST` to display a progress bar
**drill_set**

**Referen ces**

Drill documentation

**Examples**

```r
## Not run:
drill_connection()

## Not run:
drill_query("SELECT * FROM cp.`employee.json` limit 5")

## End(Not run)
```

---

**set Drill SYSTEM or SESSION options**

**Description**

Helper function to make it more R-like to set Drill SESSION or SYSTEM options. It handles the conversion of R types (like `TRUE`) to SQL types and automatically quotes parameter values (when necessary).

**Usage**

```r
drill_set(drill_con, ..., type = c("session", "system"))
```

**Arguments**

- `drill_con` : drill server connection object setup by `drill_connection()`
- `...` : named parameters to be sent to ALTER [SYSTEM|SESSION]
- `type` : set the session or system parameter

**Details**

If any query errors result, error messages will be presented to the console.

**Value**

a `tbl` (invisibly) with the ALTER queries sent and results, including errors.

**References**

Drill documentation

**Examples**

```r
## Not run:
drill_connection()

## Not run:
drill_set(exec.errors.verbose=TRUE, store.format="parquet", web.logs.max_lines=20000)

## End(Not run)
```
### drill_settings_reset

*Changes (optionally, all) session settings back to system defaults*

**Description**

Changes (optionally, all) session settings back to system defaults

**Usage**

```r
drill_settings_reset(drill_con, ...)
```

**Arguments**

- **drill_con**: drill server connection object setup by `drill_connection()`
- **...**: bare name of system options to reset

**References**

- Drill documentation

**Examples**

```r
## Not run:
drill_connection() %>% drill_settings_reset(exec.errors.verbose)

## End(Not run)
```

### drill_show_files

*Show files in a file system schema.*

**Description**

Show files in a file system schema.

**Usage**

```r
drill_show_files(drill_con, schema_spec)
```

**Arguments**

- **drill_con**: drill server connection object setup by `drill_connection()`
- **schema_spec**: properly quoted "filesystem.directory_name" reference path

**References**

- Drill documentation
**drill_show_schemas**

**Examples**

```plaintext
## Not run:
drill_connection() => drill_show_files("dfs.tmp")

## End(Not run)
```

**drill_show_schemas**  
*Returns a list of available schemas.*

**Description**

Returns a list of available schemas.

**Usage**

```plaintext
drill_show_schemas(drill_con)
```

**Arguments**

- `drill_con`  
  drill server connection object setup by `drill_connection()`

**References**

Drill documentation

---

**drill_stats**  
*Get Drillbit information, such as ports numbers*

**Description**

Get Drillbit information, such as ports numbers

**Usage**

```plaintext
drill_stats(drill_con)
```

**Arguments**

- `drill_con`  
  drill server connection object setup by `drill_connection()`

**References**

Drill documentation
Examples

```
## Not run:
drill_connection() %>% drill_stats()

## End(Not run)
```

---

**drill_status**  
*Get the status of Drill*

---

**Description**

Get the status of Drill

**Usage**

```
drill_status(drill_con)
```

**Arguments**

- **drill_con**  
drill server connection object setup by `drill_connection()`

**Note**

The output of this is in a "viewer" window

**Examples**

```
## Not run:
drill_connection() %>% drill_stats()

## End(Not run)
```

---

**drill_storage**  
*Get the list of storage plugin names and configurations*

---

**Description**

Get the list of storage plugin names and configurations

**Usage**

```
drill_storage(drill_con, plugin = NULL)
```

**Arguments**

- **drill_con**  
drill server connection object setup by `drill_connection()`
- **plugin**  
the assigned name in the storage plugin definition.
drill_system_reset

References

Drill documentation

Examples

## Not run:
drill_connection() %>% drill_storage()

## End(Not run)

---

**drill_system_reset**  *Changes (optionally, all) system settings back to system defaults*

### Description

Changes (optionally, all) system settings back to system defaults

### Usage

```
drill_system_reset(drill_con, ..., all = FALSE)
```

### Arguments

- **drill_con**: drill server connection object setup by `drill_connection()`
- **...**: bare name of system options to reset
- **all**: if TRUE, all parameters are reset (... is ignored)

### References

Drill documentation

### Examples

## Not run:
drill_connection() %>% drill_system_reset(all=TRUE)

## End(Not run)
drill_threads  Get information about threads

Description
Get information about threads

Usage
```
drill_threads(drill_con)
```

Arguments
- `drill_con`: drill server connection object setup by `drill_connection()`

Note
The output of this is in a "viewer" window

Examples
```
## Not run:
drill_connection() %>% drill_threads()

## End(Not run)
```

---

drill_uplift  Turn columnar query results into a type-converted tbl

Description
If you know the result of `drill_query()` will be a data frame, then you can pipe it to this function to pull out `rows` and automatically type-convert it.

Usage
```
drill_uplift(query_result)
```

Arguments
- `query_result`: the result of a call to `drill_query()`

Details
Not really intended to be called directly, but useful if you accidentally ran `drill_query()` without `uplift=TRUE` but want to then convert the structure.
**drill_use**

**Change to a particular schema.**

**Description**
Change to a particular schema.

**Usage**

```
drill_use(drill_con, schema_name)
```

**Arguments**

- **drill_con**: drill server connection object setup by `drill_connection()`
- **schema_name**: A unique name for a Drill schema. A schema in Drill is a configured storage plugin, such as hive, or a storage plugin and workspace.

**References**
Drill documentation

---

**drill_version**

**Identify the version of Drill running**

**Description**
Identify the version of Drill running

**Usage**

```
drill_version(drill_con)
```

**Arguments**

- **drill_con**: drill server connection object setup by `drill_connection()`

**References**
Drill documentation

**Examples**

```r
## Not run:
drill_connection() %>% drill_version()
```

## End(Not run)
Drill is an innovative low-latency distributed query engine designed to enable data exploration and analytics on both relational and non-relational datastores, scaling to petabytes of data. Users can query the data using standard SQL and BI tools without having to create and manage schemas. Some of the key features are:

- Schema-free JSON document model similar to MongoDB and Elasticsearch
- Industry-standard APIs: ANSI SQL, ODBC/JDBC, RESTful APIs
- Extremely user and developer friendly
- Pluggable architecture enables connectivity to multiple datastores

Drill includes a distributed execution environment, purpose built for large-scale data processing. At the core of Drill is the "Drillbit" service which is responsible for accepting requests from the client, processing the queries, and returning results to the client.

You can install and run a Drillbit service on one node or on many nodes to form a distributed cluster environment. When a Drillbit runs on each data node in a cluster, Drill can maximize data locality during query execution without moving data over the network or between nodes. Drill uses ZooKeeper to maintain cluster membership and health check information.

Methods are provided to work with Drill via the native JDBC & REST APIs along with R dbi and dplyr interfaces.

Author(s)

Bob Rudis (bob@rud.is)

References

Drill documentation

Description

The following functions are imported and then re-exported from the sergeant package to enable use of the magrittr pipe operator with no additional library calls
**src_drill**  

*Connect to Drill (dplyr)*

**Description**

Use `src_drill()` to connect to a Drill cluster and `tbl()` to connect to a fully-qualified "table reference". The vast majority of Drill SQL functions have also been made available to the dplyr interface. If you have custom Drill SQL functions that need to be implemented please file an issue on GitHub.

**Usage**

```r
src_drill(host = Sys.getenv("DRILL_HOST", "localhost"),
          port = as.integer(Sys.getenv("DRILL_PORT", 8047L)), ssl = FALSE)
```

```r
## S3 method for class 'src_drill'
tbl(src, from, ...)
```

**Arguments**

- **host**: Drill host (will pick up the value from DRILL_HOST env var)
- **port**: Drill port (will pick up the value from DRILL_PORT env var)
- **ssl**: use ssl?
- **src**: A Drill "src" created with `src_drill()`
- **from**: A Drill view or table specification
- **...**: Extra parameters

**Note**

This is a DBI wrapper around the Drill REST API. TODO username/password support

**Examples**

```r
## Not run:
db <- src_drill("localhost", 8047L)

print(db)
```

```r
## src: DrillConnection
## tbs: INFORMATION_SCHEMA, cp.default, dfs.default, dfs.root, dfs.tmp, sys
emp <- tbl(db, "cp.employee.json")

count(emp, gender, marital_status)
```

```r
## # Source: lazy query [?? x 3]
## # Database: DrillConnection
## # Groups: gender
## marital_status gender n
```
### src_drill

```r
## <chr>  <chr>  <int>
## 1 S     F    297
## 2 M     M    278
## 3 S     M    276

# Drill-specific SQL functions are also available
```n

```r
select(emp, full_name) %>%
  mutate(
    loc = strpos(full_name, "a"),
    first_three = substr(full_name, 1L, 3L),
    len = length(full_name),
    rx = regexp_replace(full_name, "[aeiouAEIOU]", "*")
  )
```

```r
## # Source: lazy query [?? x 9]
## # Database: DrillConnection
## # loc full_name len rpdw pos rx
## # <int>  <chr>  <int> <chr> <int> <chr>
## # 1 0 Sheri Nowmer 12 Sheri Nowmer******** 0      Sh*r* N*w*r
## # 2 0 Derrick Whelply 15 Derrick Whelply***** 0   D*r*kck Wh*elply
## # 3 5 Michael Spence 14 Michael Spence***** 11 M<ch**1 Sp*n*c*
## # 4 2 Maya Gutierrez 14 Maya Gutierrez***** 0      M*y* G*t**r**z
## # 5 7 Roberta Damstra 15 Roberta Damstra***** 0 R*b*r*t* D*mstr*
## # 6 7 Rebecca Kanagaki 16 Rebecca Kanagaki***** 0 R*b*c*c* K*n*g*k*
## # 7 0 Kim Brunner 11 Kim Brunner********** 0   K*m Br*n*n*r
## # 8 6 Brenda Blumberg 15 Brenda Blumberg***** 3 B*rnd* B*l*mb*rg
## # 9 2 Darren Stanz 12 Darren Stanz******* 5 D*r*r*n St*nz
## # 10 4 Jonathan Murraiin 17 Jonathan Murraiin***** 0 J*n*nth*n M*rr****n
## # ... with more rows, and 3 more variables: rpd <chr>, rnd <dbl>, first_three <chr>
```

```r
## # End(Not run)
```
Index

%>% (sergeant-exports), 18

dbConnect, DrillDriver-method (Drill), 3
dbDataType, DrillConnection-method, 2
dbUnloadDriver, DrillDriver-method, 3
Drill, 3
drill_active, 4
drill_cancel, 4
drill_connection, 5
drill_custom_functions, 5
drill_jdbc, 7
drill_metrics, 8
drill_options, 9
drill_profile, 9
drill_profiles, 10
drill_query, 10
drill_set, 11
drill_settings_reset, 12
drill_show_files, 12
drill_show_schemas, 13
drill_stats, 13
drill_status, 14
drill_storage, 14
drill_system_reset, 15
drill_threads, 16
drill_uplift, 16
drill_use, 17
drill_version, 17
DrillDriver, 3

sergeant, 18
sergeant-exports, 18
sergeant-package (sergeant), 18
src_drill, 19

tbl.src_drill (src_drill), 19