Package 'AzureKusto'

October 12, 2022

Title Interface to 'Kusto'/'Azure Data Explorer'

Version 1.0.7

Description An interface to 'Azure Data Explorer', also known as 'Kusto', a fast, highly scalable data exploration service from Microsoft: <https://azure.microsoft.com/en-us/services/data-explorer/>. Includes 'DBI' and 'dplyr' interfaces, with the latter modelled after the 'dbplyr' package, whereby queries are translated from R into the native 'KQL' query language and executed lazily. On the admin side, the package extends the object framework provided by 'AzureRMR' to support creation and deletion of databases, and management of database principals. Part of the 'AzureR' family of packages.

URL https://github.com/Azure/AzureKusto
https://github.com/Azure/AzureR

BugReports https://github.com/Azure/AzureKusto/issues

License MIT + file LICENSE

Encoding UTF-8

VignetteBuilder knitr

Depends R (>= 3.3)

Imports rlang, methods, utils, httr (>= 1.3), jsonlite, R6, openssl, AzureAuth, AzureRMR (>= 2.0.0), tibble, dplyr, tidyselect (>= 0.2.4), DBI (>= 1.0.0)

Suggests bit64, knitr, testthat, tidyr, AzureGraph, AzureStor (>= 2.0.0), rmarkdown

RoxygenNote 7.2.1

NeedsCompilation no

Author Hong Ooi [aut],
Alex Kyllo [aut, cre],
dbplyr development team [cph] (Original framework for dplyr/database interface),
Microsoft [cph]

Maintainer Alex Kyllo <jekyllo@microsoft.com>

Repository CRAN

Date/Publication 2022-08-26 16:14:35 UTC

R topics documented:

`add_op_join` *Append a join operation to the tbl_kusto object's ops list*

Description

Append a join operation to the tbl_kusto object's ops list

Usage

```
add_op_join(
  type,
  x,
  y,
  by = NULL,
  suffix = NULL,
  .strategy = NULL,
  .shufflekeys = NULL,
  .num_partitions = NULL,
  .remote = NULL
)
```

Arguments

`type`	The name of the join type, one of: inner_join, left_join, right_join, full_join, semi_join, anti_join
`x`	The "left" tbl
`y`	The "right" tbl
`by`	A vector of column names; keys by which tbl x and tbl y will be joined
`suffix`	A vector of strings that will be appended to the names of non-join key columns that exist in both tbl x and tbl y to distinguish them by source tbl.
`.strategy`	A strategy hint to provide to Kusto.
`.shufflekeys`	A character vector of column names to shuffle on, if `.strategy = "shuffle"`.
`.num_partitions`	The number of partitions for a shuffle query.
`.remote`	A strategy hint to provide to Kusto for cross-cluster joins.

add_op_set_op *Append a set operation to the tbl_kusto object's ops list*

Description

Append a set operation to the tbl_kusto object's ops list

Usage

```
add_op_set_op(x, y, type)
```

Arguments

x	The "left" tbl
y	The "right" tbl
type	The type of set operation to perform, currently only supports union_all

add_op_single *Append an operation representing a single-table verb to the tbl_kusto object's ops list*

Description

Append an operation representing a single-table verb to the tbl_kusto object's ops list

Usage

```
add_op_single(name, .data, dots = list(), args = list())
```

Arguments

name	The name of the operation, e.g. 'select', 'filter'
.data	The tbl_kusto object to append the operation to
dots	The expressions passed as arguments to the operation verb
args	Other non-expression arguments passed to the operation verb

AzureKusto *DBI interface: connect to a Kusto cluster*

Description

Functions to connect to a Kusto cluster.

Usage

```
AzureKusto()

## S4 method for signature 'AzureKustoDriver'
dbConnect(drv, ..., bigint = c("numeric", "integer64"))

## S4 method for signature 'AzureKustoDriver'
dbCanConnect(drv, ...)

## S4 method for signature 'AzureKustoDriver'
dbDisconnect(conn, ...)
```

Arguments

drv	An AzureKusto DBI driver object, instantiated with `AzureKusto()`.
...	Authentication arguments supplied to `kusto_database_endpoint`.
bigint	How to treat Kusto long integer columns. By default, they will be converted to R numeric variables. If this is "integer64", they will be converted to `integer64` variables using the bit64 package.
conn	For `dbDisconnect`, an AzureKustoConnection object obtained with `dbConnect`.

Details

Kusto is connectionless, so `dbConnect` simply wraps a database endpoint object, generated with `kusto_database_endpoint(...)`. The endpoint itself can be accessed via the `@endpoint` slot. Similarly, `dbDisconnect` always returns TRUE.

`dbCanConnect` attempts to detect whether querying the database with the given information and credentials will be successful. The result may not be accurate; essentially all it does is check that its arguments are valid Kusto properties. Ultimately the best way to tell if querying will work is to try it.

Value

For `dbConnect`, an object of class AzureKustoConnection.

For `dbCanConnect`, TRUE if authenticating with the Kusto server succeeded with the given arguments, and FALSE otherwise.

For `dbDisconnect`, always TRUE, invisibly.

See Also

kusto-DBI, dbReadTable, dbWriteTable, dbGetQuery, dbSendStatement, kusto_database_endpoint

Examples

```
## Not run:
db <- DBI::dbConnect(AzureKusto(),
   server="https://mycluster.westus.kusto.windows.net", database="database", tenantid="contoso")

DBI::dbDisconnect(db)

# no authentication credentials: returns FALSE
DBI::dbCanConnect(AzureKusto(),
    server="https://mycluster.westus.kusto.windows.net")


## End(Not run)
```

---

az_kusto *Kusto/Azure Data Explorer cluster resource class*

Description

Class representing a Kusto cluster, exposing methods for working with it.

Methods

The following methods are available, in addition to those provided by the AzureRMR::az_resource class:

- `new(...)`: Initialize a new storage object. See 'Initialization'.
- `start()`: Start the cluster.
- `stop()`: Stop the cluster.
- `create_database(...)`: Create a new Kusto database. See `Databases` below.
- `get_database(database))`: Get an existing database.
- `delete_database(database, confirm=TRUE)`: Delete a database, by default asking for confirmation first.
- `list_databases()`: List all databases in this cluster.
- `get_default_tenant()`: Retrieve the default tenant to authenticate with this cluster.
- `get_query_token(tenant, ...)`: Obtain an authentication token from Azure Active Directory for this cluster's query endpoint. Accepts further arguments that will be passed to get_kusto_token.
- `get_ingestion_token(tenant, ...)`: Obtain an authentication token for this cluster's ingestion endpoint. Accepts further arguments that will be passed to get_kusto_token.

Initialization

Initializing a new object of this class can either retrieve an existing Kusto cluster, or create a new cluster on the host. Generally, the best way to initialize an object is via the `get_kusto_cluster` and `create_kusto_cluster` methods of the az_resource_group class, which handle the details automatically.

Databases

A Kusto cluster can have several databases, which are represented in AzureKusto via az_kusto_database R6 objects. The `az_kusto` class provides the `create_database`, `get_database`, `delete_database` and `list_databases` methods for creating, deleting and retrieving databases. It's recommended to use these methods rather than calling `az_kusto_database$new()` directly.

`create_database` takes the following arguments. It returns an object of class az_kusto_database

- `database`: The name of the database to create.
- `retention_period`: The retention period of the database, after which data will be soft-deleted.
- `cache_period`: The cache period of the database, the length of time for which queries will be cached.

`get_database` takes a single argument `database`, the name of the database to retrieve, and returns an object of class `az_kusto_database`. `delete_database` takes the name of the database to delete and returns NULL on a successful deletion. `list_databases` takes no arguments and returns a list of `az_kusto_database` objects, one for each database in the cluster.

See Also

az_kusto_database, kusto_database_endpoint, create_kusto_cluster, get_kusto_cluster, delete_kusto_cluster, get_kusto_token

Kusto/Azure Data Explorer documentation,

Examples

```
## Not run:

# recommended way of retrieving a resource: via a resource group object
kus <- resgroup$get_kusto_cluster("mykusto")

# list databases
kust$list_databases()

# create a new database with a retention period of 6 months
kust$create_database("newdb", retention_period=180)

# get the default authentication tenant
kus$get_default_tenant()

# generate an authentication token
kust$get_aad_token()
```

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

`az_kusto_database` *Kusto/Azure Data Explorer database resource class*

Description

Class representing a Kusto database, exposing methods for working with it.

Methods

The following methods are available, in addition to those provided by the AzureRMR::az_resource class:

- `new(...)`: Initialize a new storage object. See 'Initialization'.
- `add_principals(...)`: Add new database principals. See `Principals` below.
- `remove_principals(...)`: Remove database principals.
- `list_principals()`: Retrieve all database principals, as a data frame.
- `get_query_endpoint()`: Get a query endpoint object for interacting with the database.
- `get_ingestion_endpoint()`: Get an ingestion endpoint object for interacting with the database.

Initialization

Initializing a new object of this class can either retrieve an existing Kusto database, or create a new database on the server. Generally, the best way to initialize an object is via the `get_database`, `list_databases()` and `create_database` methods of the az_kusto class, which handle the details automatically.

Principals

This class provides methods for managing the principals of a database.

`add_principal` takes the following arguments. It returns a data frame with one row per principal, containing the details for each principal.

- `name`: The name of the principal to create.
- `role`: The role of the principal, for example "Admin" or "User".
- `type`: The type of principal, either "User" or "App".
- `fqn`: The fully qualified name of the principal, for example "aaduser=username@mydomain" for an Azure Active Directory account. If supplied, the other details will be obtained from this.
- `email`: For a user principal, the email address.
- `app_id`: For an application principal, the ID.

`remove_principal` removes a principal. It takes the same arguments as `add_principal`; if the supplied details do not match the actual details for the principal, it is not removed.

See Also

az_kusto, kusto_database_endpoint, create_database, get_database, delete_database

Kusto/Azure Data Explorer documentation,

Examples

```
## Not run:

# recommended way of retrieving a resource: via a resource group object
db <- resgroup$
    get_kusto_cluster("mykusto")$
    get_database("mydatabase")

# list principals
db$list_principals()

# add a new principal
db$add_principal("New User", role="User", fqn="aaduser=username@mydomain")

# get the endpoint
db$get_database_endpoint(use_integer64=FALSE)


## End(Not run)
```

`base_agg` *Aggregation function translations*

Description

Aggregation function translations

Usage

```
base_agg
```

Format

An object of class `environment` of length 7.

base_scalar *Scalar operator translations (infix and prefix)*

Description

Scalar operator translations (infix and prefix)

Usage

```
base_scalar
```

Format

An object of class `environment` of length 75.

base_window *Window function translations*

Description

Window function translations

Usage

```
base_window
```

Format

An object of class `environment` of length 1.

build_kql *Build a KQL string.*

Description

Build a KQL string.

Usage

```
build_kql(..., .env = parent.frame())
```

Arguments

`...`	input to convert to KQL. Use `kql()` to preserve user input as is (dangerous), and `ident()` to label user input as kql identifiers (safe)
`.env`	the environment in which to evaluate the arguments. Should not be needed in typical use.

`collect.tbl_kusto` *Compile the preceding dplyr operations into a kusto query, execute it on the remote server, and return the result as a tibble.*

Description

Compile the preceding dplyr operations into a kusto query, execute it on the remote server, and return the result as a tibble.

Usage

```
## S3 method for class 'tbl_kusto'
collect(tbl, ...)
```

Arguments

`tbl`	An instance of class tbl_kusto representing a Kusto table
`...`	needed for agreement with generic. Not otherwise used.

`compute.tbl_kusto` *Execute the query, store the results in a table, and return a reference to the new table*

Description

Execute the query, store the results in a table, and return a reference to the new table

Usage

```
## S3 method for class 'tbl_kusto'
compute(tbl, name = generate_table_name(), ...)
```

Arguments

`tbl`	An instance of class tbl_kusto representing a Kusto table
`name`	The name for the Kusto table to be created. If name is omitted, the table will be named Rtbl_ + 8 random lowercase letters
`...`	other parameters passed to the query

`copy_to.kusto_database_endpoint` *This function uploads a local data frame into a remote data source, creating the table definition as needed. If the table exists, it will append the data to the existing table. If not, it will create a new table.*

Description

This function uploads a local data frame into a remote data source, creating the table definition as needed. If the table exists, it will append the data to the existing table. If not, it will create a new table.

Usage

```
## S3 method for class 'kusto_database_endpoint'
copy_to(
  dest,
  df,
  name = deparse(substitute(df)),
  overwrite = FALSE,
  method = "inline",
  ...
)
```

Arguments

`dest`	remote data source
`df`	local data frame
`name`	Name for new remote table
`overwrite`	If `TRUE`, will overwrite an existing table with name `name`. If `FALSE`, will throw an error if `name` already exists.
`method`	For local ingestion, the method to use. "inline", "streaming", or "indirect".
`...`	other parameters passed to the query

See Also

`collect()` for the opposite action; downloading remote data into a local tbl.

`create_kusto_cluster` *Create Kusto/Azure Data Explorer cluster*

Description

Method for the AzureRMR::az_resource_group class.

Usage

```
create_kusto_cluster(name, location,
                     node_size="D14_v2", ...)
```

Arguments

- `name`: The name of the cluster.
- `location`: The location/region in which to create the account. Defaults to the resource group location.
- `node_size`: The capacity of the nodes in each of the cluster. Defaults to "D14_v2", which should be available in all regions. The availability of other sizes depends on the region the cluster is created in.
- ... Other named arguments to pass to the az_kusto initialization function.

Details

This method deploys a new Kusto cluster resource, with parameters given by the arguments.

Value

An object of class `az_kusto` representing the created cluster.

See Also

get_kusto_cluster, delete_kusto_cluster, az_kusto

Kusto/Azure Data Explorer documentation

Examples

```
## Not run:

rg <- AzureRMR::get_azure_login("myaadtenant")$
    get_subscription("subscription_id")$
    get_resource_group("rgname")

# create a new Kusto cluster
rg$create_kusto_cluster("mykusto", node_size="L16")


## End(Not run)
```

dbGetQuery,AzureKustoConnection,character-method
DBI methods for Kusto queries and commands

Description

DBI methods for Kusto queries and commands

Usage

```
## S4 method for signature 'AzureKustoConnection,character'
dbGetQuery(conn, statement, ...)

## S4 method for signature 'AzureKustoConnection'
dbSendQuery(conn, statement, ...)

## S4 method for signature 'AzureKustoResult'
dbFetch(res, n = -1, ...)

## S4 method for signature 'AzureKustoConnection,character'
dbSendStatement(conn, statement, ...)

## S4 method for signature 'AzureKustoConnection,character'
dbExecute(conn, statement, ...)

## S4 method for signature 'AzureKustoConnection,character'
dbListFields(conn, name, ...)

## S4 method for signature 'AzureKustoResult'
dbColumnInfo(res, ...)
```

Arguments

conn	An AzureKustoConnection object.
statement	A string containing a Kusto query or control command.
...	Further arguments passed to run_query.
res	An AzureKustoResult resultset object
n	The number of rows to return. Not used.
name	For dbListFields, a table name.

Details

These are the basic DBI functions to query the database. Note that Kusto only supports synchronous queries and commands; in particular, dbSendQuery and dbSendStatement will wait for the query or statement to complete, rather than returning immediately.

dbSendStatement and dbExecute are meant for running Kusto control commands, and will throw an error if passed a regular query. dbExecute also returns the entire result of running the command, rather than simply a row count.

See Also

dbConnect, dbReadTable, dbWriteTable, run_query

Examples

```
## Not run:

db <- DBI::dbConnect(AzureKusto(),
    server="https://mycluster.location.kusto.windows.net", database="database"...)

DBI::dbGetQuery(db, "iris | count")
DBI::dbListFields(db, "iris")

# does the same thing as dbGetQuery, but returns an AzureKustoResult object
res <- DBI::dbSendQuery(db, "iris | count")
DBI::dbFetch(res)
DBI::dbColumnInfo(res)

DBI::dbExecute(db, ".show tables")

# does the same thing as dbExecute, but returns an AzureKustoResult object
res <- DBI::dbSendStatement(db, ".show tables")
DBI::dbFetch(res)


## End(Not run)
```

---

dbReadTable,AzureKustoConnection,character-method

DBI methods for Kusto table management

Description

DBI methods for Kusto table management

Usage

```
## S4 method for signature 'AzureKustoConnection,character'
dbReadTable(conn, name, ...)

## S4 method for signature 'AzureKustoConnection,ANY'
dbWriteTable(conn, name, value, method, ...)

## S4 method for signature 'AzureKustoConnection'
```

```
dbCreateTable(conn, name, fields, ..., row.names = NULL, temporary = FALSE)

## S4 method for signature 'AzureKustoConnection,ANY'
dbRemoveTable(conn, name, ...)

## S4 method for signature 'AzureKustoConnection'
dbListTables(conn, ...)

## S4 method for signature 'AzureKustoConnection,ANY'
dbExistsTable(conn, name, ...)
```

Arguments

conn	An AzureKustoConnection object.
name	A string containing a table name.
...	Further arguments passed to run_query.
value	For dbWriteTable, a data frame to be written to a Kusto table.
method	For dbWriteTable, the ingestion method to use to write the table. See ingest_local.
fields	For dbCreateTable, the table specification: either a named character vector, or a data frame of sample values.
row.names	For dbCreateTable, the row names. Not used.
temporary	For dbCreateTable, whether to create a temporary table. Must be FALSE for Kusto.

Details

These functions read, write, create and delete a table, list the tables in a Kusto database, and check for table existence. With the exception of dbWriteTable, they ultimately call run_query which does the actual work of communicating with the Kusto server. dbWriteTable calls ingest_local to write the data to the server; note that it only supports ingesting a local data frame, as per the DBI spec.

Kusto does not have the concept of temporary tables, so calling dbCreateTable with temporary set to anything other than FALSE will generate an error.

dbReadTable and dbWriteTable are likely to be of limited use in practical scenarios, since Kusto tables tend to be much larger than available memory.

Value

For dbReadTable, an in-memory data frame containing the table.

See Also

AzureKusto-connection, dbConnect, run_query, ingest_local

Examples

```
## Not run:
db <- DBI::dbConnect(AzureKusto(),
    server="https://mycluster.location.kusto.windows.net", database="database"...)

DBI::dbListTables(db)

if(!DBI::dbExistsTable(db, "mtcars"))
    DBI::dbCreateTable(db, "mtcars")

DBI::dbWriteTable(db, "mtcars", mtcars, method="inline")

DBI::dbReadTable(db, "mtcars")

DBI::dbRemoveTable(db, "mtcars")


## End(Not run)
```

`delete_kusto_cluster` *Delete Kusto/Azure Data Explorer cluster*

Description

Method for the AzureRMR::az_resource_group class.

Usage

```
delete_kusto_cluster(name, confirm=TRUE, wait=FALSE)
```

Arguments

- `name`: The name of the cluster.
- `confirm`: Whether to ask for confirmation before deleting.
- `wait`: Whether to wait until the deletion is complete.

Value

NULL on successful deletion.

See Also

create_kusto_cluster, get_kusto_cluster, az_kusto

Kusto/Azure Data Explorer documentation

Examples

```
## Not run:

rg <- AzureRMR::az_rm$
    new(tenant="myaadtenant.onmicrosoft.com", app="app_id", password="password")$
    get_subscription("subscription_id")$
    get_resource_group("rgname")

# delete a Kusto cluster
rg$delete_kusto_cluster("mycluster")


## End(Not run)
```

`escape` *Escape/quote a string.*

Description

Escape/quote a string.

Usage

```
escape(x, parens = NA, collapse = " ")

kql_vector(x, parens = NA, collapse = " ")
```

Arguments

`x`	An object to escape. Existing kql vectors will be left as is, character vectors are escaped with single quotes, numeric vectors have trailing `.0` added if they're whole numbers, identifiers are escaped with double quotes.
`parens, collapse`	Controls behaviour when multiple values are supplied. `parens` should be a logical flag, or if `NA`, will wrap in parens if length > 1. Default behaviour: lists are always wrapped in parens and separated by commas, identifiers are separated by commas and never wrapped, atomic vectors are separated by spaces and wrapped in parens if needed.

flatten_query — *Walks the tree of ops and builds a stack.*

Description

Walks the tree of ops and builds a stack.

Usage

```
flatten_query(op, ops = list())
```

Arguments

op	the current operation
ops	the stack of operations to append to, recursively

get_kusto_cluster — *Get existing Kusto/Azure Data Explorer cluster*

Description

Method for the AzureRMR::az_resource_group class.

Usage

```
get_kusto_cluster(name, location,
                      node_size="D14_v2")
```

Arguments

- name: The name of the cluster.

Details

This method retrieves an existing Kusto cluster resource.

Value

An object of class az_kusto representing the created cluster.

See Also

create_kusto_cluster, delete_kusto_cluster, az_kusto

Kusto/Azure Data Explorer documentation

Examples

```
## Not run:

rg <- AzureRMR::get_azure_login("myaadtenant")$
    get_subscription("subscription_id")$
    get_resource_group("rgname")

# get a Kusto cluster
rg$get_kusto_cluster("mykusto")


## End(Not run)
```

---

get_kusto_token *Manage AAD authentication tokens for Kusto clusters*

Description

Manage AAD authentication tokens for Kusto clusters

Usage

```
get_kusto_token(
  server = NULL,
  clustername,
  location = NULL,
  tenant = NULL,
  app = .kusto_app_id,
  auth_type = NULL,
  version = 2,
  ...
)

delete_kusto_token(
  server = NULL,
  clustername,
  location = NULL,
  tenant = NULL,
  app = .kusto_app_id,
  auth_type = NULL,
  version = 2,
  ...,
  hash = NULL,
  confirm = TRUE
)

list_kusto_tokens()
```

Arguments

server	The URI of your Kusto cluster. If not supplied, it is obtained from the clustername and location arguments.
clustername	The cluster name.
location	The cluster location. Leave this blank for a Microsoft-internal Kusto cluster like "help".
tenant	Your Azure Active Directory (AAD) tenant. Can be a GUID, a name ("myaadtenant") or a fully qualified domain name ("myaadtenant.com").
app	The ID of the Azure Active Directory app/service principal to authenticate with. Defaults to the ID of the KustoClient app.
auth_type	The authentication method to use. Can be one of "authorization_code", "device_code", "client_credentials" or "resource_owner". The default is to pick one based on the other arguments.
version	The AAD version to use. There should be no reason to change this from the default value of 2.
...	Other arguments to pass to AzureAuth::get_azure_token.
hash	For delete_kusto_token, the MD5 hash of the token. This is used to identify the token if provided.
confirm	For delete_kusto_token, whether to ask for confirmation before deleting the token.

Details

get_kusto_token returns an authentication token for the given cluster, caching its value on disk. delete_kusto_token deletes a cached token, and list_kusto_tokens lists all cached tokens.

By default, authentication tokens will be obtained using the main KustoClient Active Directory app. This app can be used to authenticate with any Kusto cluster (assuming, of course, you have the proper credentials).

Value

get_kusto_token returns an object of class AzureAuth::AzureToken representing the authentication token, while list_kusto_tokens returns a list of such objects. delete_azure_token returns NULL on a successful delete.

See Also

kusto_database_endpoint, AzureAuth::get_azure_token

Examples

```
## Not run:

get_kusto_token("https://myclust.australiaeast.kusto.windows.net")
get_kusto_token(clustername="myclust", location="australiaeast")
```

```
# authenticate using client_credentials method: see ?AzureAuth::get_azure_token
get_kusto_token("https://myclust.australiaeast.kusto.windows.net",
                tenant="mytenant", app="myapp", password="password")


## End(Not run)
```

`ident` *Flag a character string as a Kusto identifier*

Description

Flag a character string as a Kusto identifier

Usage

```
ident(...)
```

Arguments

`...`	character strings to flag as Kusto identifiers

`ident_q` *Pass an already-escaped string to Kusto*

Description

Pass an already-escaped string to Kusto

Usage

```
ident_q(...)
```

Arguments

`...`	character strings to treat as already-escaped identifiers

ingest_local *Ingestion functions for Kusto*

Description

Ingestion functions for Kusto

Usage

```
ingest_local(
  database,
  src,
  dest_table,
  method = NULL,
  staging_container = NULL,
  ingestion_token = database$token,
  http_status_handler = "stop",
  ...
)

ingest_url(database, src, dest_table, async = FALSE, ...)

ingest_blob(
  database,
  src,
  dest_table,
  async = FALSE,
  key = NULL,
  token = NULL,
  sas = NULL,
  ...
)

ingest_adls2(
  database,
  src,
  dest_table,
  async = FALSE,
  key = NULL,
  token = NULL,
  sas = NULL,
  ...
)

ingest_adls1(
  database,
  src,
```

```
  dest_table,
  async = FALSE,
  key = NULL,
  token = NULL,
  sas = NULL,
  ...
)
```

Arguments

database	A Kusto database endpoint object, created with kusto_database_endpoint.
src	The source data. This can be either a data frame, local filename, or URL.
dest_table	The name of the destination table.
method	For local ingestion, the method to use. See 'Details' below.
staging_container	For local ingestion, an Azure storage container to use for staging the dataset. This can be an object of class either AzureStor::blob_container or AzureStor::adls_filesystem. Only used if method="indirect".
ingestion_token	For local ingestion, an Azure Active Directory authentication token for the cluster ingestion endpoint. Only used if method="streaming".
http_status_handler	For local ingestion, how to handle HTTP conditions >= 300. Defaults to "stop"; alternatives are "warn", "message" and "pass". The last option will pass through the raw response object from the server unchanged, regardless of the status code. This is mostly useful for debugging purposes, or if you want to see what the Kusto REST API does. Only used if method="streaming".
...	Named arguments to be treated as ingestion parameters.
async	For the URL ingestion functions, whether to do the ingestion asychronously. If TRUE, the function will return immediately while the server handles the operation in the background.
key, token, sas	Authentication arguments for the Azure storage ingestion methods. If multiple arguments are supplied, a key takes priority over a token, which takes priority over a SAS. Note that these arguments are for authenticating with the Azure *storage account*, as opposed to Kusto itself.

Details

There are up to 3 possible ways to ingest a local dataset, specified by the method argument.

- method="indirect": The data is uploaded to blob storage, and then ingested from there. This is the default if the AzureStor package is present.
- method="streaming": The data is uploaded to the cluster ingestion endpoint. This is the default if the AzureStor package is not present, however be aware that currently (as of February 2019) streaming ingestion is in beta and has to be enabled for a cluster by filing a support ticket.

- method="inline": The data is embedded into the command text itself. This is only recommended for testing purposes, or small datasets.

Note that the destination table must be created ahead of time for the ingestion to proceed.

Examples

```
## Not run:

# ingesting from local:

# ingest via Azure storage
cont <- AzureStor::storage_container("https://mystorage.blob.core.windows.net/container",
    sas="mysas")
ingest_local(db, "file.csv", "table",
    method="indirect", storage_container=cont)

ingest_local(db, "file.csv", "table", method="streaming")

# ingest by inlining data into query
ingest_inline(db, "file.csv", "table", method="inline")

# ingesting online data:

# a public dataset: Microsoft web data from UCI machine learning repository
ingest_url(db,
   "https://archive.ics.uci.edu/ml/machine-learning-databases/anonymous/anonymous-msweb.data",
    "table")

# from blob storage:
ingest_blob(db,
    "https://mystorage.blob.core.windows.net/container/myblob",
    "table",
    sas="mysas")

# from ADLSGen2:
token <- AzureRMR::get_azure_token("https://storage.azure.com", "mytenant", "myapp", "password")
ingest_blob(db,
    "abfss://filesystem@myadls2.dfs.core.windows.net/data/myfile",
    "table",
    token=token)


## End(Not run)
```

---

inner_join.tbl_kusto_abstract

Join methods for Kusto tables

Description

These methods are the same as other joining methods, with the exception of the `.strategy`, `.shufflekeys` and `.num_partitions` optional arguments. They provide hints to the Kusto engine on how to execute the join, and can sometimes be useful to speed up a query. See the Kusto documentation for more details.

Usage

```
## S3 method for class 'tbl_kusto_abstract'
inner_join(
  x,
  y,
  by = NULL,
  suffix = c(".x", ".y"),
  .strategy = NULL,
  .shufflekeys = NULL,
  .num_partitions = NULL,
  .remote = NULL,
  ...
)

## S3 method for class 'tbl_kusto_abstract'
left_join(
  x,
  y,
  by = NULL,
  suffix = c(".x", ".y"),
  .strategy = NULL,
  .shufflekeys = NULL,
  .num_partitions = NULL,
  .remote = NULL,
  ...
)

## S3 method for class 'tbl_kusto_abstract'
right_join(
  x,
  y,
  by = NULL,
  suffix = c(".x", ".y"),
  .strategy = NULL,
  .shufflekeys = NULL,
  .num_partitions = NULL,
  .remote = NULL,
  ...
)

## S3 method for class 'tbl_kusto_abstract'
```

```
full_join(
  x,
  y,
  by = NULL,
  suffix = c(".x", ".y"),
  .strategy = NULL,
  .shufflekeys = NULL,
  .num_partitions = NULL,
  .remote = NULL,
  ...
)

## S3 method for class 'tbl_kusto_abstract'
semi_join(
  x,
  y,
  by = NULL,
  suffix = c(".x", ".y"),
  .strategy = NULL,
  .shufflekeys = NULL,
  .num_partitions = NULL,
  .remote = NULL,
  ...
)

## S3 method for class 'tbl_kusto_abstract'
anti_join(
  x,
  y,
  by = NULL,
  suffix = c(".x", ".y"),
  .strategy = NULL,
  .shufflekeys = NULL,
  .num_partitions = NULL,
  .remote = NULL,
  ...
)
```

Arguments

x, y	Kusto tbls.
by	The columns to join on.
suffix	The suffixes to use for deduplicating column names.
.strategy	A join strategy hint to pass to Kusto. Currently the values supported are "shuffle" and "broadcast".
.shufflekeys	A character vector of column names to use as shuffle keys.
.num_partitions	The number of partitions for a shuffle query.

.remote	A join strategy hint to use for cross-cluster joins. Can be "left", "right", "local" or "auto" (the default).
...	Other arguments passed to lower-level functions.

See Also

dplyr::join

Examples

```
## Not run:

tbl1 <- tbl_kusto(db, "table1")
tbl2 <- tbl_kusto(db, "table2")

# standard dplyr syntax:
left_join(tbl1, tbl2)

# Kusto extensions:
left_join(tbl1, tbl2, .strategy="broadcast") # a broadcast join

left_join(tbl1, tbl2, .shufflekeys=c("var1", "var2")) # shuffle join with shuffle keys

left_join(tbl1, tbl2, .num_partitions=5)     # no. of partitions for a shuffle join

## End(Not run)
```

---

is_kusto_database *Information functions*

Description

These functions test whether an object is of the given class.

Usage

```
is_kusto_database(x)

is_kusto_cluster(x)
```

Arguments

x	An R object.

kql — *Tag character strings as Kusto Query Language. Assumes the string is valid and properly escaped.*

Description

Tag character strings as Kusto Query Language. Assumes the string is valid and properly escaped.

Usage

```
kql(...)
```

Arguments

...	character strings to tag as KQL

kql_aggregate — *Return a function representing a KQL aggregation function*

Description

Return a function representing a KQL aggregation function

Usage

```
kql_aggregate(f)
```

Arguments

f	Name of the Kusto aggregation function

kql_build — *Build the tbl object into a data structure representing a Kusto query*

Description

Build the tbl object into a data structure representing a Kusto query

Usage

```
kql_build(op)
```

Arguments

op	A nested sequence of query operations, i.e. tbl_kusto$ops

`kql_build.op_mutate` *dplyr's mutate verb can include aggregations, but Kusto's extend verb cannot. If the mutate contains no aggregations, then it can emit an extend clause. If the mutate contains an aggregation and the tbl is ungrouped, then it must emit a summarize clause grouped by all variables. If the mutate contains an aggregation and the tbl is grouped, then it must join to a subquery containing the summarize clause.*

Description

dplyr's mutate verb can include aggregations, but Kusto's extend verb cannot. If the mutate contains no aggregations, then it can emit an extend clause. If the mutate contains an aggregation and the tbl is ungrouped, then it must emit a summarize clause grouped by all variables. If the mutate contains an aggregation and the tbl is grouped, then it must join to a subquery containing the summarize clause.

Usage

```
## S3 method for class 'op_mutate'
kql_build(op, ...)
```

Arguments

`op`	A nested sequence of query operations, i.e. tbl_kusto$ops
`...`	Needed for agreement with generic. Not otherwise used.

`kql_escape_ident` *Escape a Kusto identifier with [' ']*

Description

Escape a Kusto identifier with [' ']

Usage

```
kql_escape_ident(x)
```

Arguments

`x`	An identifier to escape

kql_escape_ident_q *Pass through an already-escaped Kusto identifier*

Description

Pass through an already-escaped Kusto identifier

Usage

```
kql_escape_ident_q(x)
```

Arguments

x	An identifier to pass through

kql_escape_logical *Escape a Kusto logical value. Converts TRUE/FALSE to true / false*

Description

Escape a Kusto logical value. Converts TRUE/FALSE to true / false

Usage

```
kql_escape_logical(x)
```

Arguments

x	A logical value to escape

kql_escape_string *Escape a Kusto string by single-quoting*

Description

Escape a Kusto string by single-quoting

Usage

```
kql_escape_string(x)
```

Arguments

x	A string to escape

kql_infix *Return a function representing a scalar KQL infix operator*

Description

Return a function representing a scalar KQL infix operator

Usage

```
kql_infix(f)
```

Arguments

f	Name of a Kusto infix operator / function

kql_prefix *Return a function representing a scalar KQL prefix function*

Description

Return a function representing a scalar KQL prefix function

Usage

```
kql_prefix(f, n = NULL)
```

Arguments

f	Name of a Kusto infix function
n	Number of arguments accepted by the Kusto prefix function

kql_render *Render a set of operations on a tbl_kusto_abstract to a Kusto query*

Description

Render a set of operations on a tbl_kusto_abstract to a Kusto query

Usage

```
kql_render(query, ...)
```

Arguments

query	The tbl_kusto instance with a sequence of operations in $ops
...	needed for agreement with generic. Not otherwise used.

kql_translate_env *Build a kql_variant class out of the environments holding scalar and aggregation function definitions*

Description

Build a kql_variant class out of the environments holding scalar and aggregation function definitions

Usage

```
kql_translate_env()
```

kql_translator *Builds an environment from a list of R -> Kusto query language translation pairs.*

Description

Builds an environment from a list of R -> Kusto query language translation pairs.

Usage

```
kql_translator(..., .funs = list(), .parent = new.env(parent = emptyenv()))
```

Arguments

...	Pairs of R call = Kusto call translations as individual arguments
.funs	Parse of R call = Kusto call translations in list format
.parent	A parent environment to attach this env onto

kql_window *Return a function representing a KQL window function*

Description

Return a function representing a KQL window function

Usage

```
kql_window(f)
```

Arguments

f	Name of the Kusto aggregation function

kusto-DBI *DBI interface to Kusto*

Description

AzureKusto implements a subset of the DBI specification for interfacing with databases in R. The following methods are supported:

- Connections: dbConnect, dbDisconnect, dbCanConnect
- Table management: dbExistsTable, dbCreateTable, dbRemoveTable, dbReadTable, dbWriteTable
- Querying: dbGetQuery, dbSendQuery, dbFetch, dbSendStatement, dbExecute, dbListFields, dbColumnInfo

Details

Kusto is quite different to the SQL databases that DBI targets, which affects the behaviour of certain DBI methods and renders other moot.

- Kusto is connectionless. `dbConnect` simply wraps a database endpoint object, created with kusto_database_endpoint. Similarly, `dbDisconnect` always returns TRUE. `dbCanConnect` attempts to check if querying the database will succeed, but this may not be accurate.
- Temporary tables are not a Kusto concept, so `dbCreateTable(*, temporary=TRUE)` will throw an error.
- It only supports synchronous queries, with a default timeout of 4 minutes. `dbSendQuery` and `dbSendStatement` will wait for the query to execute, rather than returning immediately. The object returned contains the full result of the query, which `dbFetch` extracts.
- The Kusto Query Language (KQL) is not SQL, and so higher-level SQL methods are not implemented.

kusto_database_endpoint *Endpoints for communicating with a Kusto database*

Description

Endpoints for communicating with a Kusto database

Usage

```
kusto_database_endpoint(
  ...,
  .connection_string = NULL,
  .query_token = NULL,
  .use_integer64 = FALSE
)
```

Arguments

`...`	Named arguments which are the properties for the endpoint object. See 'Details' below for the properties that AzureKusto recognises.
`.connection_string`	An alternative way of specifying the properties, as a database connection string. Properties supplied here override those in `...` if they overlap.
`.query_token`	Optionally, an Azure Active Directory (AAD) token to authenticate with. If this is supplied, it overrides other tokens specified in `...` or in the connection string.
`.use_integer64`	For `kusto_database_endpoint`, whether to convert columns with Kusto `long` datatype into 64-bit integers in R, using the bit64 package. If FALSE, represent them as numeric instead.

Details

This is a list of properties recognised by `kusto_database_endpoint`, and their alternate names. Property names not in this list will generate an error. Note that not all properties that are recognised are currently supported by AzureKusto.

General properties:

- server: The URI of the server, usually of the form 'https://clustername.location.kusto.windows.net'.
  - addr, address, network address, datasource, host
- database: The database.
  - initialcatalog, dbname
- tenantid: The AAD tenant name or ID to authenticate with.
  - authority
- appclientid: The AAD app/service principal ID
  - applicationclientid
- traceclientversion: The client version for tracing.
- queryconsistency: The level of query consistency. Defaults to "weakconsistency".
- response_dynamic_serialization: How to serialize dynamic responses.
- response_dynamic_serialization_2: How to serialize dynamic responses.

User authentication properties:

- password
- user: The user name.
  - uid, userid
- traceusername: The user name for tracing.
- usertoken: The AAD token for user authentication.
-
  - usertoken, usrtoken
- fed: Logical, whether federated authentication is enabled. Currently unsupported; if this is TRUE, `kusto_database_endpoint` will print a warning and ignore it.

  - federated security, federated, aadfed, aadfederatedsecurity

App authentication properties:

- appkey: The secret key for the app.
  - applicationkey
- traceappname: The AAD app for tracing.
- apptoken: The AAD token for app authentication.
  - apptoken, applicationtoken

Currently, AzureKusto only supports authentication via Azure Active Directory. Authenticating with DSTS is planned for the future.

The way `kusto_database_endpoint` obtains an AAD token is as follows.

1. If the `.query_token` argument is supplied, use it.
2. Otherwise, if the `usertoken` property is supplied, use it.
3. Otherwise, if the `apptoken` property is supplied, use it.
4. Otherwise, if the `appclientid` property is supplied, use it to obtain a token:
   - With the `user` and `pwd` properties if available
   - Or with the `appkey` property if available
   - Otherwise do an interactive authentication and ask for the user credentials
5. Otherwise, if no `appclientid` property is supplied, authenticate with the KustoClient app:
   - With the `user` and `pwd` properties if available
   - Otherwise do an interactive authentication and ask for the user credentials using a device code

Value

An object of class `kusto_database_endpoint`.

See Also

run_query, az_kusto_database

Examples

```
## Not run:

kusto_database_endpoint(server="myclust.australiaeast.kusto.windows.net", database="db1")

# supplying a token obtained previously
token <- get_kusto_token("myclust.australiaeast.kusto.windows.net")
kusto_database_endpoint(server="myclust.australiaeast.kusto.windows.net", database="db1",
                        .query_token=token)


## End(Not run)
```

nest.tbl_kusto_abstract *Nest method for Kusto tables*

Description

This method collapses a column into a list

Usage

```
nest.tbl_kusto_abstract(.data, ...)
```

Arguments

.data	A kusto tbl.
...	Specification of columns to nest. Translates to summarize make_list() in Kusto.

op_base *The "base case" operation representing the tbl itself and its column variables*

Description

The "base case" operation representing the tbl itself and its column variables

Usage

```
op_base(x, vars, class = character())
```

Arguments

x	A tbl object
vars	A vector of column variables in the tbl
class	The class that op_base should inherit from, default is character()

op_double *A double-table verb, e.g. joins, setops*

Description

A double-table verb, e.g. joins, setops

Usage

```
op_double(name, x, y, args = list())
```

Arguments

name	The name of the operation, e.g. 'left_join', 'union_all'
x	The "left" tbl
y	The "right" tbl
args	Other arguments passed to the operation verb

op_grps *Look up the applicable grouping variables for an operation based on the data source and preceding sequence of operations*

Description

Look up the applicable grouping variables for an operation based on the data source and preceding sequence of operations

Usage

```
op_grps(op)
```

Arguments

op	An operation instance

op_single *A class representing a single-table verb*

Description

A class representing a single-table verb

Usage

```
op_single(name, x, dots = list(), args = list())
```

Arguments

name	the name of the operation verb, e.g. "select", "filter"
x	the tbl object
dots	expressions passed to the operation verb function
args	other arguments passed to the operation verb function

op_vars *Look up the applicable variables in scope for a given operation based on the data source and preceding sequence of operations*

Description

Look up the applicable variables in scope for a given operation based on the data source and preceding sequence of operations

Usage

```
op_vars(op)
```

Arguments

op	An operation instance

`run_query` *Run a query or command against a Kusto database*

Description

Run a query or command against a Kusto database

Usage

```
run_query(database, qry_cmd, ..., .http_status_handler = "stop")
```

Arguments

`database`	A Kusto database endpoint object, as returned by `kusto_database_endpoint`.
`qry_cmd`	A string containing the query or command. In KQL, a database management command is a statement that starts with a "."
`...`	Named arguments to be used as parameters for a parameterized query. These are ignored for database management commands.
`.http_status_handler`	The function to use to handle HTTP status codes. The default "stop" will throw an R error via `httr::stop_for_status` if the status code is not less than 300; other possibilities are "warn", "message" and "pass". The last option will pass through the raw response object from the server unchanged, regardless of the status code. This is mostly useful for debugging purposes, or if you want to see what the Kusto REST API does.

Details

This function is the workhorse of the AzureKusto package. It communicates with the Kusto server and returns the query or command results, as data frames.

See Also

kusto_database_endpoint, ingest_local, ingest_url, ingest_blob, ingest_adls2

Examples

```
## Not run:

endp <- kusto_database_endpoint(server="myclust.australiaeast.kusto.windows.net", database="db1")

# a command
run_query(endp, ".show table iris")

# a query
run_query(endp, "iris | count")


## End(Not run)
```

show_query.tbl_kusto_abstract

Translate a sequence of dplyr operations on a tbl into a Kusto query string.

Description

Translate a sequence of dplyr operations on a tbl into a Kusto query string.

Usage

```
## S3 method for class 'tbl_kusto_abstract'
show_query(tbl)
```

Arguments

tbl	A tbl_kusto or tbl_kusto_abstract instance

summarise.tbl_kusto_abstract

Summarise method for Kusto tables

Description

This method is the same as other summarise methods, with the exception of the `.strategy`, `.shufflekeys` and `.num_partitions` optional arguments. They provide hints to the Kusto engine on how to execute the summarisation, and can sometimes be useful to speed up a query. See the Kusto documentation for more details.

Usage

```
## S3 method for class 'tbl_kusto_abstract'
summarise(
  .data,
  ...,
  .strategy = NULL,
  .shufflekeys = NULL,
  .num_partitions = NULL
)
```

Arguments

.data	A Kusto tbl.
...	Summarise expressions.
.strategy	A summarise strategy to pass to Kusto. Currently the only value supported is "shuffle".
.shufflekeys	A character vector of column names to use as shuffle keys.
.num_partitions	The number of partitions for a shuffle query.

See Also

dplyr::summarise

Examples

```
## Not run:

tbl1 <- tbl_kusto(db, "table1")

## standard dplyr syntax:
summarise(tbl1, mx=mean(x))

## Kusto extensions:
summarise(tbl1, mx=mean(x), .strategy="broadcast") # a broadcast summarise

summarise(tbl1, mx=mean(x), .shufflekeys=c("var1", "var2")) # shuffle summarise with shuffle keys

summarise(tbl1, mx=mean(x), .num_partitions=5)    # no. of partitions for a shuffle summarise

## End(Not run)
```

---

tbl_kusto *A tbl object representing a table in a Kusto database.*

Description

A tbl object representing a table in a Kusto database.

Usage

```
tbl_kusto(kusto_database, table_name, ...)
```

Arguments

kusto_database	An instance of kusto_database_endpoint that this table should be queried from
table_name	The name of the table in the Kusto database
...	parameters to pass in case the Kusto source table is a parameterized function.

translate_kql *Translate R expressions into Kusto Query Language equivalents.*

Description

Translate R expressions into Kusto Query Language equivalents.

Usage

```
translate_kql(...)
```

Arguments

...	Expressions to translate.

unnest.tbl_kusto_abstract *Unnest method for Kusto tables*

Description

This method takes a list column and expands it so that each element of the list gets its own row. unnest() translates to Kusto's mv-expand operator.

Usage

```
unnest.tbl_kusto_abstract(.data, ..., .id = NULL)
```

Arguments

.data	A Kusto tbl.
...	Specification of columns to unnest.
.id	Data frame identifier - if supplied, will create a new column with name .id, giving a unique identifier. This is most useful if the list column is named.

Index