Package 'elastic'

June 26, 2018

Title General Purpose Interface to 'Elasticsearch'

Description Connect to 'Elasticsearch', a 'NoSQL' database built on the 'Java' Virtual Machine. Interacts with the 'Elasticsearch' 'HTTP' API (<https://www.elastic.co/products/elasticsearch>), including functions for setting connection details to 'Elasticsearch' instances, loading bulk data, searching for documents with both 'HTTP' query variables and 'JSON' based body requests. In addition, 'elastic' provides functions for interacting with API's for 'indices', documents, nodes, clusters, an interface to the cat API, and more.

Version 0.8.4

License MIT + file LICENSE

URL https://github.com/ropensci/elastic

BugReports https://github.com/ropensci/elastic/issues

VignetteBuilder knitr

Imports utils, methods, httr (>= 1.2.1), curl (>= 2.2), jsonlite (>= 1.1)

Suggests roxygen2 (>= 6.0.1), knitr, testthat

RoxygenNote 6.0.1

X-schema.org-applicationCategory Databases

X-schema.org-keywords database, Elasticsearch, HTTP, API, search, NoSQL, Java, JSON, documents

X-schema.org-isPartOf https://ropensci.org

NeedsCompilation no

Author Scott Chamberlain [aut, cre]

Maintainer Scott Chamberlain <myrmecocystus@gmail.com>

Repository CRAN

Date/Publication 2018-06-26 05:50:14 UTC

R topics documented:

alias *Elasticsearch alias APIs*

Description

Elasticsearch alias APIs

Usage

```
alias_get(index = NULL, alias = NULL, ignore_unavailable = FALSE, ...)

aliases_get(index = NULL, alias = NULL, ignore_unavailable = FALSE, ...)

alias_exists(index = NULL, alias = NULL, ...)

alias_create(index = NULL, alias, routing = NULL, filter = NULL, ...)

alias_delete(index = NULL, alias, ...)
```

Arguments

index	An index name
alias	An alias name
ignore_unavailable	(logical) What to do if an specified index name doesn't exist. If set to TRUE then those indices are ignored.
...	Curl args passed on to `httr::POST()`
routing	Ignored for now
filter	Ignored for now

Author(s)

Scott Chamberlain myrmecocystus@gmail.com

References

https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-aliases.html

Examples

```
## Not run:
# Create/update an alias
alias_create(index = "plos", alias = "tables")

# Retrieve a specified alias
alias_get(index="plos")
```

```
alias_get(alias="tables")
aliases_get()

# Check for alias existence
alias_exists(index = "plos")
alias_exists(alias = "tables")
alias_exists(alias = "adsfasdf")

# Delete an alias
alias_delete(index = "plos", alias = "tables")
alias_exists(alias = "tables")

# Curl options
library("httr")
alias_create(index = "plos", alias = "tables")
aliases_get(alias = "tables", config=verbose())

## End(Not run)
```

---

cat *Use the cat Elasticsearch api.*

Description

Use the cat Elasticsearch api.

Usage

```
cat_(parse = FALSE, ...)

cat_aliases(verbose = FALSE, index = NULL, h = NULL, help = FALSE,
  bytes = FALSE, parse = FALSE, ...)

cat_allocation(verbose = FALSE, h = NULL, help = FALSE, bytes = FALSE,
  parse = FALSE, ...)

cat_count(verbose = FALSE, index = NULL, h = NULL, help = FALSE,
  bytes = FALSE, parse = FALSE, ...)

cat_segments(verbose = FALSE, index = NULL, h = NULL, help = FALSE,
  bytes = FALSE, parse = FALSE, ...)

cat_health(verbose = FALSE, h = NULL, help = FALSE, bytes = FALSE,
  parse = FALSE, ...)

cat_indices(verbose = FALSE, index = NULL, h = NULL, help = FALSE,
  bytes = FALSE, parse = FALSE, ...)
```

```
cat_master(verbose = FALSE, index = NULL, h = NULL, help = FALSE,
  bytes = FALSE, parse = FALSE, ...)

cat_nodes(verbose = FALSE, h = NULL, help = FALSE, bytes = FALSE,
  parse = FALSE, ...)

cat_nodeattrs(verbose = FALSE, h = NULL, help = FALSE, bytes = FALSE,
  parse = FALSE, ...)

cat_pending_tasks(verbose = FALSE, h = NULL, help = FALSE,
  bytes = FALSE, parse = FALSE, ...)

cat_plugins(verbose = FALSE, h = NULL, help = FALSE, bytes = FALSE,
  parse = FALSE, ...)

cat_recovery(verbose = FALSE, index = NULL, h = NULL, help = FALSE,
  bytes = FALSE, parse = FALSE, ...)

cat_thread_pool(verbose = FALSE, index = NULL, h = NULL, help = FALSE,
  bytes = FALSE, parse = FALSE, ...)

cat_shards(verbose = FALSE, index = NULL, h = NULL, help = FALSE,
  bytes = FALSE, parse = FALSE, ...)

cat_fielddata(verbose = FALSE, index = NULL, fields = NULL, h = NULL,
  help = FALSE, bytes = FALSE, parse = FALSE, ...)
```

Arguments

parse	(logical) Parse to a data.frame or not. Default: FALSE
...	Curl args passed on to httr::GET()
verbose	(logical) If TRUE (default) the url call used printed to console
index	(character) Index name
h	(character) Fields to return
help	(logical) Output available columns, and their meanings
bytes	(logical) Give numbers back machine friendly. Default: FALSE
fields	(character) Fields to return, only used with fielddata

Details

See https://www.elastic.co/guide/en/elasticsearch/reference/current/cat.html for the cat API documentation.

Note how cat_() has an underscore at the end to avoid conflict with the function base::cat() in base R.

Examples

```
## Not run:
# list Elasticsearch cat endpoints
cat_()

# Do other cat operations
cat_aliases()
cat_aliases(index='plos')
cat_allocation()
cat_allocation(verbose=TRUE)
cat_count()
cat_count(index='plos')
cat_count(index='gbif')
cat_segments()
cat_segments(index='gbif')
cat_health()
cat_indices()
cat_master()
cat_nodes()
# cat_nodeattrs() # not available in older ES versions
cat_pending_tasks()
cat_plugins()
cat_recovery(verbose=TRUE)
cat_recovery(index='gbif')
cat_thread_pool()
cat_thread_pool(verbose=TRUE)
cat_shards()
cat_fielddata()
cat_fielddata(fields='body')

# capture cat data into a data.frame
cat_(parse = TRUE)
cat_indices(parse = TRUE)
cat_indices(parse = TRUE, verbose = TRUE)
cat_count(parse = TRUE)
cat_count(parse = TRUE, verbose = TRUE)
cat_health(parse = TRUE)
cat_health(parse = TRUE, verbose = TRUE)

# Get help - what does each column mean
head(cat_indices(help = TRUE, parse = TRUE))
cat_health(help = TRUE, parse = TRUE)
head(cat_nodes(help = TRUE, parse = TRUE))

# Get back only certain fields
cat_nodes()
cat_nodes(h = c('ip','port','heapPercent','name'))
cat_nodes(h = c('id', 'ip', 'port', 'v', 'm'))
cat_indices(verbose = TRUE)
cat_indices(verbose = TRUE, h = c('index','docs.count','store.size'))

# Get back machine friendly numbers instead of the normal human friendly
```

```
cat_indices(verbose = TRUE, bytes = TRUE)

# Curl options
library("httr")
cat_count(config=verbose())

## End(Not run)
```

`cluster` *Elasticsearch cluster endpoints*

Description

Elasticsearch cluster endpoints

Usage

```
cluster_settings(index = NULL, raw = FALSE, callopts = list(),
  verbose = TRUE, ...)

cluster_health(index = NULL, level = NULL, wait_for_status = NULL,
  wait_for_relocating_shards = NULL, wait_for_active_shards = NULL,
  wait_for_nodes = NULL, timeout = NULL, raw = FALSE, callopts = list(),
  verbose = TRUE, ...)

cluster_state(index = NULL, metrics = NULL, raw = FALSE,
  callopts = list(), verbose = TRUE, ...)

cluster_stats(index = NULL, raw = FALSE, callopts = list(),
  verbose = TRUE, ...)

cluster_reroute(body, raw = FALSE, callopts = list(), ...)

cluster_pending_tasks(index = NULL, raw = FALSE, callopts = list(),
  verbose = TRUE, ...)
```

Arguments

`index`	Index
`raw`	If `TRUE` (default), data is parsed to list. If `FALSE`, then raw JSON.
`callopts`	Curl args passed on to `httr::POST()`
`verbose`	If `TRUE` (default) the url call used printed to console.
`...`	Further args passed on to elastic search HTTP API as parameters.
`level`	Can be one of cluster, indices or shards. Controls the details level of the health information returned. Defaults to cluster.

`wait_for_status`
: One of green, yellow or red. Will wait (until the timeout provided) until the status of the cluster changes to the one provided or better, i.e. green > yellow > red. By default, will not wait for any status.

`wait_for_relocating_shards`
: A number controlling to how many relocating shards to wait for. Usually will be 0 to indicate to wait till all relocations have happened. Defaults to not wait.

`wait_for_active_shards`
: A number controlling to how many active shards to wait for. Defaults to not wait.

`wait_for_nodes`
: The request waits until the specified number N of nodes is available. It also accepts >=N, <=N, >N and <N. Alternatively, it is possible to use ge(N), le(N), gt(N) and lt(N) notation.

`timeout`
: A time based parameter controlling how long to wait if one of the wait_for_XXX are provided. Defaults to 30s.

`metrics`
: One or more of version, master_node, nodes, routing_table, metadata, and blocks. See Details.

`body`
: Query, either a list or json.

Details

metrics param options:

- version Shows the cluster state version.
- master_node Shows the elected master_node part of the response
- nodes Shows the nodes part of the response
- routing_table Shows the routing_table part of the response. If you supply a comma separated list of indices, the returned output will only contain the indices listed.
- metadata Shows the metadata part of the response. If you supply a comma separated list of indices, the returned output will only contain the indices listed.
- blocks Shows the blocks part of the response

Additional parameters that can be passed in:

- metric A comma-separated list of metrics to display. Possible values: '_all', 'completion', 'docs', 'fielddata', 'filter_cache', 'flush', 'get', 'id_cache', 'indexing', 'merge', 'percolate', 'refresh', 'search', 'segments', 'store', 'warmer'
- completion_fields A comma-separated list of fields for completion metric (supports wildcards)
- fielddata_fields A comma-separated list of fields for fielddata metric (supports wildcards)
- fields A comma-separated list of fields for fielddata and completion metric (supports wildcards)
- groups A comma-separated list of search groups for search statistics
- allow_no_indices Whether to ignore if a wildcard indices expression resolves into no concrete indices. (This includes _all string or when no indices have been specified)

- expand_wildcards Whether to expand wildcard expression to concrete indices that are open, closed or both.
- ignore_indices When performed on multiple indices, allows to ignore missing ones (default: none)
- ignore_unavailable Whether specified concrete indices should be ignored when unavailable (missing or closed)
- human Whether to return time and byte values in human-readable format.
- level Return stats aggregated at cluster, index or shard level. ('cluster', 'indices' or 'shards', default: 'indices')
- types A comma-separated list of document types for the indexing index metric

Examples

```
## Not run:
cluster_settings()
cluster_health()

cluster_state()
cluster_state(metrics = "version")
cluster_state(metrics = "nodes")
cluster_state(metrics = c("version", "nodes"))
cluster_state(metrics = c("version", "nodes", 'blocks'))
cluster_state("shakespeare", metrics = "metadata")
cluster_state(c("shakespeare", "flights"), metrics = "metadata")

cluster_stats()
cluster_pending_tasks()

body <- '{
  "commands" : [ {
    "move" :
      {
        "index" : "test", "shard" : 0,
        "from_node" : "node1", "to_node" : "node2"
      }
    },
    {
      "allocate" : {
          "index" : "test", "shard" : 1, "node" : "node3"
      }
    }
  ]
}'
# cluster_reroute(body =  body)

cluster_health()
# cluster_health(wait_for_status = "yellow", timeout = "3s")

## End(Not run)
```

connect *Set connection details to an Elasticsearch engine.*

Description

Set connection details to an Elasticsearch engine.

Usage

```
connect(es_host = "127.0.0.1", es_port = 9200, es_path = NULL,
  es_transport_schema = "http", es_user = NULL, es_pwd = NULL,
  force = FALSE, errors = "simple", es_base = NULL, headers = NULL, ...)

connection()
```

Arguments

es_host	(character) The base host, defaults to 127.0.0.1 Synonym of es_base
es_port	(character) port to connect to, defaults to 9200 (optional)
es_path	(character) context path that is appended to the end of the url. Default: NULL, ignored
es_transport_schema	(character) http or https. Default: http
es_user	(character) User name, if required for the connection. You can specify, but ignored for now.
es_pwd	(character) Password, if required for the connection. You can specify, but ignored for now.
force	(logical) Force re-load of connection details
errors	(character) One of simple (Default) or complete. Simple gives http code and error message on an error, while complete gives both http code and error message, and stack trace, if available.
es_base	(character) Synonym of es_host, and will be gone in a future version of **elastic**
headers	Either an object of class request or a list that can be coerced to an object of class request via httr::add_headers(). These headers are used in all requests. To use headers in individual requests and not others, pass in headers using httr::add_headers() via ... in a function call.
...	Further args passed on to print for the es_conn class.

Details

The default configuration is set up for localhost access on port 9200, with no username or password.

connect() and connection() no longer ping the Elasticsearch server, but only print your connection details.

Internally, we store your connection settings with environment variables. That means you can set your env vars permanently in .Renviron file, and use them on a server e.g., as private env vars

See Also

`ping()` to check your connection

Examples

```
## Not run:
# the default is set to 127.0.0.1 (i.e., localhost) and port 9200
connect()

# set a different host
# connect(es_host = '162.243.152.53')
# => http://162.243.152.53:9200

# set a different port
# connect(es_port = 8000)
# => http://localhost:8000

# set a different context path
# connect(es_path = 'foo_bar')
# => http://localhost:9200/foo_bar

# set to https
# connect(es_transport_schema = 'https')
# => https://localhost:9200

# See connection details
connection()

# set headers
connect(headers = list(a = 5))
## or
connect(headers = add_headers(a = 5))

## End(Not run)
```

---

count *Get counts of the number of records per index.*

Description

Get counts of the number of records per index.

Usage

```
count(index = NULL, type = NULL, callopts = list(), verbose = TRUE, ...)
```

Arguments

index	Index, defaults to all indices
type	Document type
callopts	Curl args passed on to httr::GET.
verbose	If TRUE (default) the url call used printed to console.
...	Further args passed on to elastic search HTTP API as parameters.

Details

See docs for the count API here https://www.elastic.co/guide/en/elasticsearch/reference/current/search-count.html

You can also get a count of documents using Search() or Search_uri() and setting size = 0

Examples

```
## Not run:
count()
count(index='plos')
count(index='plos', type='article')
count(index='shakespeare')
count(index=c('plos','shakespeare'), q="a*")
count(index=c('plos','shakespeare'), q="z*")

# Curl options
library("httr")
count(callopts=verbose())

## End(Not run)
```

---

docs_bulk *Use the bulk API to create, index, update, or delete documents.*

Description

Use the bulk API to create, index, update, or delete documents.

Usage

```
docs_bulk(x, index = NULL, type = NULL, chunk_size = 1000,
  doc_ids = NULL, es_ids = TRUE, raw = FALSE, quiet = FALSE, ...)
```

Arguments

x	A list, data.frame, or character path to a file. required.
index	(character) The index name to use. Required for data.frame input, but optional for file inputs.
type	(character) The type name to use. If left as NULL, will be same name as index.
chunk_size	(integer) Size of each chunk. If your data.frame is smaller thank chunk_size, this parameter is essentially ignored. We write in chunks because at some point, depending on size of each document, and Elasticsearch setup, writing a very large number of documents in one go becomes slow, so chunking can help. This parameter is ignored if you pass a file name. Default: 1000
doc_ids	An optional vector (character or numeric/integer) of document ids to use. This vector has to equal the size of the documents you are passing in, and will error if not. If you pass a factor we convert to character. Default: not passed
es_ids	(boolean) Let Elasticsearch assign document IDs as UUIDs. These are sequential, so there is order to the IDs they assign. If TRUE, doc_ids is ignored. Default: TRUE
raw	(logical) Get raw JSON back or not. If TRUE you get JSON; if FALSE you get a list. Default: FALSE
quiet	(logical) Suppress progress bar. Default: FALSE
...	Pass on curl options to httr::POST()

Details

More on the Bulk API: https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-bulk.html

This function dispatches on data.frame or character input. Character input has to be a file name or the function stops with an error message.

If you pass a data.frame to this function, we by default to an index operation, that is, create the record in the index and type given by those parameters to the function. Down the road perhaps we will try to support other operations on the bulk API. if you pass a file, of course in that file, you can specify any operations you want.

Row names are dropped from data.frame, and top level names for a list are dropped as well.

A progress bar gives the progress for data.frames and lists - the progress bar is based around a for loop, where progress indicates progress along the iterations of the for loop, where each iteration is a chunk of data that's converted to bulk format, then pushed into Elasticsearch. The character method has no for loop, so no progress bar.

Value

A list

Document IDs

Document IDs can be passed in via the `doc_ids` paramater when passing in data.frame or list, but not with files. If ids are not passed to `doc_ids`, we assign document IDs from 1 to length of the object (rows of a data.frame, or length of a list). In the future we may allow the user to select whether they want to assign sequential numeric IDs or to allow Elasticsearch to assign IDs, which are UUIDs that are actually sequential, so you still can determine an order of your documents.

Document IDs and Factors

If you pass in ids that are of class factor, we coerce them to character with `as.character`. This applies to both data.frame and list inputs, but not to file inputs.

Large numbers for document IDs

Until recently, if you had very large integers for document IDs, `docs_bulk` failed. It should be fixed now. Let us know if not.

Missing data

As of **elastic** version `0.7.8.9515` we convert `NA` to `null` before loading into Elasticsearch. Previously, fields that had an `NA` were dropped - but when you read data back from Elasticsearch into R, you retain those missing values as **jsonlite** fills those in for you. Now, fields with `NA`'s are made into `null`, and are not dropped in Elasticsearch.

Note also that null values can not be indexed or searched https://www.elastic.co/guide/en/elasticsearch/reference/5.3/null-value.html

Tips

This function returns the response from Elasticsearch, but you'll likely not be that interested in the response. If not, wrap your call to `docs_bulk` in `invisible()`, like so: `invisible(docs_bulk(...))`

Connections/Files

We create temporary files, and connections to those files, when data.frame's and lists are passed in to `docs_bulk()` (not when a file is passed in since we don't need to create a file). After inserting data into your Elasticsearch instance, we close the connections and delete the temporary files.

There are some exceptions though. When you pass in your own file, whether a tempfile or not, we don't delete those files after using them - in case you need those files again. Your own tempfile's will be cleaned up/delete when the R session ends. Non-tempfile's won't be cleaned up/deleted after the R session ends.

See Also

`docs_bulk_prep()` for prepping a newline delimited JSON file that you can load into Elasticsearch yourself. See `docs_bulk_update()` for updating documents from an R data.frame or list.

Examples

```
## Not run:
# From a file already in newline delimited JSON format
plosdat <- system.file("examples", "plos_data.json", package = "elastic")
docs_bulk(plosdat)
aliases_get()
index_delete(index='plos')
aliases_get()

# From a data.frame
docs_bulk(mtcars, index = "hello", type = "world")
## field names cannot contain dots
names(iris) <- gsub("\\.", "_", names(iris))
docs_bulk(iris, "iris", "flowers")
## type can be missing, but index can not
docs_bulk(iris, "flowers")
## big data.frame, 53K rows, load ggplot2 package first
# res <- docs_bulk(diamonds, "diam")
# Search("diam")$hits$total

# From a list
docs_bulk(apply(iris, 1, as.list), index="iris", type="flowers")
docs_bulk(apply(USArrests, 1, as.list), index="arrests")
# dim_list <- apply(diamonds, 1, as.list)
# out <- docs_bulk(dim_list, index="diamfromlist")

# When using in a loop
## We internally get last _id counter to know where to start on next bulk
## insert but you need to sleep in between docs_bulk calls, longer the
## bigger the data is
files <- c(system.file("examples", "test1.csv", package = "elastic"),
           system.file("examples", "test2.csv", package = "elastic"),
           system.file("examples", "test3.csv", package = "elastic"))
for (i in seq_along(files)) {
  d <- read.csv(files[[i]])
  docs_bulk(d, index = "testes", type = "docs")
  Sys.sleep(1)
}
count("testes", "docs")
index_delete("testes")

# You can include your own document id numbers
## Either pass in as an argument
index_create("testes")
files <- c(system.file("examples", "test1.csv", package = "elastic"),
           system.file("examples", "test2.csv", package = "elastic"),
           system.file("examples", "test3.csv", package = "elastic"))
tt <- vapply(files, function(z) NROW(read.csv(z)), numeric(1))
ids <- list(1:tt[1],
           (tt[1] + 1):(tt[1] + tt[2]),
           (tt[1] + tt[2] + 1):sum(tt))
for (i in seq_along(files)) {
```

```
  d <- read.csv(files[[i]])
  docs_bulk(d, index = "testes", type = "docs", doc_ids = ids[[i]],
    es_ids = FALSE)
}
count("testes", "docs")
index_delete("testes")

## or include in the input data
### from data.frame's
index_create("testes")
files <- c(system.file("examples", "test1_id.csv", package = "elastic"),
           system.file("examples", "test2_id.csv", package = "elastic"),
           system.file("examples", "test3_id.csv", package = "elastic"))
readLines(files[[1]])
for (i in seq_along(files)) {
  d <- read.csv(files[[i]])
  docs_bulk(d, index = "testes", type = "docs")
}
count("testes", "docs")
index_delete("testes")

### from lists via file inputs
index_create("testes")
for (i in seq_along(files)) {
  d <- read.csv(files[[i]])
  d <- apply(d, 1, as.list)
  docs_bulk(d, index = "testes", type = "docs")
}
count("testes", "docs")
index_delete("testes")

# data.frame's with a single column
## this didn't use to work, but now should work
db <- paste0(sample(letters, 10), collapse = "")
index_create(db)
res <- data.frame(foo = 1:10)
out <- docs_bulk(x = res, index = db)
count(db)
index_delete(db)



# Curl options
library("httr")
plosdat <- system.file("examples", "plos_data.json", package = "elastic")
docs_bulk(plosdat, config=verbose())


# suppress progress bar
x <- docs_bulk(mtcars, index = "hello", type = "world", quiet = TRUE)
## vs.
x <- docs_bulk(mtcars, index = "hello", type = "world", quiet = FALSE)
```

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

`docs_bulk_prep` *Use the bulk API to prepare bulk format data*

Description

Use the bulk API to prepare bulk format data

Usage

```
docs_bulk_prep(x, index, path, type = NULL, chunk_size = 1000,
  doc_ids = NULL, quiet = FALSE)
```

Arguments

`x`	A data.frame or a list. required.
`index`	(character) The index name. required.
`path`	(character) Path to the file. If data is broken into chunks, we'll use this path as the prefix, and suffix each file path with a number. required.
`type`	(character) The type name to use. If left as NULL, will be same name as index.
`chunk_size`	(integer) Size of each chunk. If your data.frame is smaller thank `chunk_size`, this parameter is essentially ignored. We write in chunks because at some point, depending on size of each document, and Elasticsearch setup, writing a very large number of documents in one go becomes slow, so chunking can help. This parameter is ignored if you pass a file name. Default: 1000
`doc_ids`	An optional vector (character or numeric/integer) of document ids to use. This vector has to equal the size of the documents you are passing in, and will error if not. If you pass a factor we convert to character. Default: not passed
`quiet`	(logical) Suppress progress bar. Default: `FALSE`

Value

File path(s). By default we use temporary files; these are cleaned up at the end of a session

Tempfiles

In `docs_bulk` we create temporary files in some cases, and delete those before the function exits. However, we don't clean up those files in this function because the point of the function is to create the newline delimited JSON files that you need. Tempfiles are cleaned up when you R session ends though - be aware of that. If you want to keep the files make sure to move them outside of the temp directory.

See Also

`docs_bulk()`

Examples

```
## Not run:
# From a data.frame
ff <- tempfile(fileext = ".json")
docs_bulk_prep(mtcars, index = "hello", type = "world", path = ff)
readLines(ff)

## field names cannot contain dots
names(iris) <- gsub("\\.", "_", names(iris))
docs_bulk_prep(iris, "iris", "flowers", path = tempfile(fileext = ".json"))

## type can be missing, but index can not
docs_bulk_prep(iris, "flowers", path = tempfile(fileext = ".json"))

# From a list
docs_bulk_prep(apply(iris, 1, as.list), index="iris", type="flowers",
   path = tempfile(fileext = ".json"))
docs_bulk_prep(apply(USArrests, 1, as.list), index="arrests",
   path = tempfile(fileext = ".json"))

# when chunking
## multiple files created, one for each chunk
bigiris <- do.call("rbind", replicate(30, iris, FALSE))
docs_bulk_prep(bigiris, index = "big", path = tempfile(fileext = ".json"))

# When using in a loop
## We internally get last _id counter to know where to start on next bulk
## insert but you need to sleep in between docs_bulk_prep calls, longer the
## bigger the data is
files <- c(system.file("examples", "test1.csv", package = "elastic"),
           system.file("examples", "test2.csv", package = "elastic"),
           system.file("examples", "test3.csv", package = "elastic"))
paths <- vector("list", length = length(files))
for (i in seq_along(files)) {
  d <- read.csv(files[[i]])
  paths[i] <- docs_bulk_prep(d, index = "stuff", type = "docs",
     path = tempfile(fileext = ".json"))
}
unlist(paths)

# You can include your own document id numbers
## Either pass in as an argument
files <- c(system.file("examples", "test1.csv", package = "elastic"),
           system.file("examples", "test2.csv", package = "elastic"),
           system.file("examples", "test3.csv", package = "elastic"))
tt <- vapply(files, function(z) NROW(read.csv(z)), numeric(1))
ids <- list(1:tt[1],
           (tt[1] + 1):(tt[1] + tt[2]),
           (tt[1] + tt[2] + 1):sum(tt))
paths <- vector("list", length = length(files))
for (i in seq_along(files)) {
  d <- read.csv(files[[i]])
```

```
  paths[i] <- docs_bulk_prep(d, index = "testes", type = "docs",
    doc_ids = ids[[i]], es_ids = FALSE, path = tempfile(fileext = ".json"))
}
unlist(paths)

## or include in the input data
### from data.frame's
files <- c(system.file("examples", "test1_id.csv", package = "elastic"),
           system.file("examples", "test2_id.csv", package = "elastic"),
           system.file("examples", "test3_id.csv", package = "elastic"))
paths <- vector("list", length = length(files))
for (i in seq_along(files)) {
  d <- read.csv(files[[i]])
  paths[i] <- docs_bulk_prep(d, index = "testes", type = "docs",
     path = tempfile(fileext = ".json"))
}
unlist(paths)

### from lists via file inputs
paths <- vector("list", length = length(files))
for (i in seq_along(files)) {
  d <- read.csv(files[[i]])
  d <- apply(d, 1, as.list)
  paths[i] <- docs_bulk_prep(d, index = "testes", type = "docs",
      path = tempfile(fileext = ".json"))
}
unlist(paths)


# suppress progress bar
docs_bulk_prep(mtcars, index = "hello", type = "world",
  path = tempfile(fileext = ".json"), quiet = TRUE)
## vs.
docs_bulk_prep(mtcars, index = "hello", type = "world",
  path = tempfile(fileext = ".json"), quiet = FALSE)

## End(Not run)
```

`docs_bulk_update` *Use the bulk API to update documents*

Description

Use the bulk API to update documents

Usage

```
docs_bulk_update(x, index = NULL, type = NULL, chunk_size = 1000,
  doc_ids = NULL, raw = FALSE, ...)
```

Arguments

x	A list, data.frame, or character path to a file. required.
index	(character) The index name to use. Required for data.frame input, but optional for file inputs.
type	(character) The type name to use. If left as NULL, will be same name as index.
chunk_size	(integer) Size of each chunk. If your data.frame is smaller thank chunk_size, this parameter is essentially ignored. We write in chunks because at some point, depending on size of each document, and Elasticsearch setup, writing a very large number of documents in one go becomes slow, so chunking can help. This parameter is ignored if you pass a file name. Default: 1000
doc_ids	An optional vector (character or numeric/integer) of document ids to use. This vector has to equal the size of the documents you are passing in, and will error if not. If you pass a factor we convert to character. Default: not passed
raw	(logical) Get raw JSON back or not. If TRUE you get JSON; if FALSE you get a list. Default: FALSE
...	Pass on curl options to httr::POST()

Details

- doc_as_upsert - is set to TRUE for all records

For doing updates with a file already prepared for the bulk API, see docs_bulk()

Only data.frame's are supported for now.

References

https://www.elastic.co/guide/en/elasticsearch/reference/6.2/docs-bulk.html#bulk-update

See Also

docs_bulk() docs_bulk_prep()

Examples

```
## Not run:
connect()
if (index_exists("foobar")) index_delete("foobar")

df <- data.frame(name = letters[1:3], size = 1:3, id = 100:102)
invisible(docs_bulk(df, 'foobar', 'foobar', es_ids = FALSE))

# add new rows in existing fields
(df2 <- data.frame(size = c(45, 56), id = 100:101))
Search("foobar", asdf = TRUE)$hits$hits
invisible(docs_bulk_update(df2, index = 'foobar', type = 'foobar'))
Search("foobar", asdf = TRUE)$hits$hits

# add new fields (and new rows by extension)
```

```
(df3 <- data.frame(color = c("blue", "red", "green"), id = 100:102))
Search("foobar", asdf = TRUE)$hits$hits
invisible(docs_bulk_update(df3, index = 'foobar', type = 'foobar'))
Search("foobar", asdf = TRUE)$hits$hits

## End(Not run)
```

`docs_create` *Create a document*

Description

Create a document

Usage

```
docs_create(index, type, id = NULL, body, version = NULL,
  version_type = NULL, op_type = NULL, routing = NULL, parent = NULL,
  timestamp = NULL, ttl = NULL, refresh = NULL, timeout = NULL,
  callopts = list(), ...)
```

Arguments

`index`	(character) The name of the index. Required
`type`	(character) The type of the document. Required
`id`	(numeric/character) The document ID. Can be numeric or character. Optional. if not provided, Elasticsearch creates the ID for you as a UUID.
`body`	The document.
`version`	(character) Explicit version number for concurrency control
`version_type`	(character) Specific version type. One of internal, external, external_gte, or force
`op_type`	(character) Operation type. One of create, or ...
`routing`	(character) Specific routing value
`parent`	(numeric) A parent document ID
`timestamp`	(date) Explicit timestamp for the document
`ttl`	(aka "time to live") Expiration time for the document. Expired documents will be expunged automatically. The expiration date that will be set for a document with a provided ttl is relative to the timestamp of the document, meaning it can be based on the time of indexing or on any time provided. The provided ttl must be strictly positive and can be a number (in milliseconds) or any valid time value (e.g, 86400000, 1d).
`refresh`	(logical) Refresh the index after performing the operation
`timeout`	(character) Explicit operation timeout, e.g,. 5m (for 5 minutes)
`callopts`	Curl options passed on to `httr::PUT()`
`...`	Further args to query DSL

References

https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-index_.html

Examples

```
## Not run:
connect()
if (!index_exists('plos')) {
  plosdat <- system.file("examples", "plos_data.json", package = "elastic")
  invisible(docs_bulk(plosdat))
}

# give a document id
x <- docs_create(index='plos', type='article', id=1002,
  body=list(id="12345", title="New title"))
x
# and the document is there now
docs_get(index='plos', type='article', id=1002)

# let Elasticsearch create the document id for you
x <- docs_create(index='plos', type='article',
  body=list(id="6789", title="Some title"))
x
# and the document is there now
docs_get(index='plos', type='article', id=x$`_id`)

## End(Not run)
```

---

docs_delete *Delete a document*

Description

Delete a document

Usage

```
docs_delete(index, type, id, refresh = NULL, routing = NULL,
  timeout = NULL, version = NULL, version_type = NULL,
  callopts = list(), ...)
```

Arguments

index	(character) The name of the index. Required
type	(character) The type of the document. Required
id	(numeric/character) The document ID. Can be numeric or character. Required
refresh	(logical) Refresh the index after performing the operation
routing	(character) Specific routing value

timeout	(character) Explicit operation timeout, e.g,. 5m (for 5 minutes)
version	(character) Explicit version number for concurrency control
version_type	(character) Specific version type. One of internal or external
callopts	Curl args passed on to `httr::DELETE()`
...	Further args to query DSL

References

https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-delete.html

Examples

```
## Not run:
if (!index_exists("plos")) {
 plosdat <- system.file("examples", "plos_data.json", package = "elastic")
 docs_bulk(plosdat)
}

# delete a document
if (!docs_get(index='plos', type='article', id=36, exists=TRUE)) {
  docs_create(index='plos', type='article', id=36,
    body = list(id="12345", title="New title")
  )
}
docs_get(index='plos', type='article', id=36)
docs_delete(index='plos', type='article', id=36)
# docs_get(index='plos', type='article', id=36) # and the document is gone

## End(Not run)
```

---

docs_get *Get documents*

Description

Get documents

Usage

```
docs_get(index, type, id, source = NULL, fields = NULL, exists = FALSE,
  raw = FALSE, callopts = list(), verbose = TRUE, ...)
```

Arguments

index	(character) The name of the index. Required
type	(character) The type of the document. Required
id	(numeric/character) The document ID. Can be numeric or character. Required
source	(logical) If TRUE, return source.
fields	Fields to return from the response object.
exists	(logical) Only return a logical as to whether the document exists or not.
raw	If TRUE (default), data is parsed to list. If FALSE, then raw JSON.
callopts	Curl args passed on to httr::POST()
verbose	If TRUE (default) the url call used printed to console.
...	Further args passed on to elastic search HTTP API as parameters.

References

https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-get.html

Examples

```
## Not run:
docs_get(index='shakespeare', type='line', id=10)
docs_get(index='shakespeare', type='line', id=12)
docs_get(index='shakespeare', type='line', id=12, source=TRUE)

# Get certain fields
if (gsub("\\.", "", ping()$version$number) < 500) {
  ### ES < v5
  docs_get(index='shakespeare', type='line', id=10, fields='play_name')
  docs_get(index='shakespeare', type='line', id=10,
    fields=c('play_name','speaker'))
} else {
  ### ES > v5
  docs_get(index='shakespeare', type='line', id=10, source='play_name')
  docs_get(index='shakespeare', type='line', id=10,
    source=c('play_name','speaker'))
}

# Just test for existence of the document
docs_get(index='plos', type='article', id=1, exists=TRUE)
docs_get(index='plos', type='article', id=123456, exists=TRUE)

## End(Not run)
```

docs_mget *Get multiple documents via the multiple get API.*

Description

Get multiple documents via the multiple get API.

Usage

```
docs_mget(index = NULL, type = NULL, ids = NULL, type_id = NULL,
  index_type_id = NULL, source = NULL, fields = NULL, raw = FALSE,
  callopts = list(), verbose = TRUE, ...)
```

Arguments

index	Index. Required.
type	Document type. Required.
ids	More than one document id, see examples.
type_id	List of vectors of length 2, each with an element for type and id.
index_type_id	List of vectors of length 3, each with an element for index, type, and id.
source	(logical) If TRUE, return source.
fields	Fields to return from the response object.
raw	If TRUE (default), data is parsed to list. If FALSE, then raw JSON.
callopts	Curl args passed on to httr::POST.
verbose	If TRUE (default) the url call used printed to console.
...	Further args passed on to elastic search HTTP API as parameters.

Details

You can pass in one of three combinations of parameters:

- Pass in something for index, type, and id. This is the simplest, allowing retrieval from the same index, same type, and many ids.
- Pass in only index and type_id - this allows you to get multiple documents from the same index, but from different types.
- Pass in only index_type_id - this is so that you can get multiple documents from different indexes and different types.

References

https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-multi-get.html

Examples

```
## Not run:
connect()

if (!index_exists('plos')) {
  plosdat <- system.file("examples", "plos_data.json", package = "elastic")
  invisible(docs_bulk(plosdat))
}

# Same index and type
docs_mget(index="plos", type="article", ids=c(9,10))

tmp <- docs_mget(index="plos", type="article", ids=c(9, 10),
  raw=TRUE)
es_parse(tmp)
docs_mget(index="plos", type="article", ids=c(9, 10),
  source='title')
docs_mget(index="plos", type="article", ids=c(14, 19),
  source=TRUE)

# curl options
library("httr")
docs_mget(index="plos", type="article", ids=1:2, callopts=verbose())

# Same index, but different types
if (!index_exists('shakespeare')) {
  shakedat <- system.file("examples", "shakespeare_data.json", package = "elastic")
  invisible(docs_bulk(shakedat))
}

docs_mget(index="shakespeare", type_id=list(c("scene",1), c("line",20)))
docs_mget(index="shakespeare", type_id=list(c("scene",1), c("line",20)),
  source='play_name')

# Different indices and different types pass in separately
docs_mget(index_type_id = list(
  c("shakespeare", "line", 20),
  c("plos", "article", 1)
 )
)

## End(Not run)
```

`docs_update` *Update a document*

Description

Update a document

Usage

```
docs_update(index, type, id, body, fields = NULL, source = NULL,
  version = NULL, version_type = NULL, routing = NULL, parent = NULL,
  timestamp = NULL, ttl = NULL, refresh = NULL, timeout = NULL,
  retry_on_conflict = NULL, wait_for_active_shards = NULL,
  detect_noop = NULL, callopts = list(), ...)
```

Arguments

index	(character) The name of the index. Required
type	(character) The type of the document. Required
id	(numeric/character) The document ID. Can be numeric or character. Required
body	The document, either a list or json
fields	A comma-separated list of fields to return in the response
source	Allows to control if and how the updated source should be returned in the response. By default the updated source is not returned. See http://bit.ly/2efmYiE filtering for details
version	(character) Explicit version number for concurrency control
version_type	(character) Specific version type. One of internal, external, external_gte, or force
routing	(character) Specific routing value
parent	ID of the parent document. Is is only used for routing and when for the upsert request
timestamp	(date) Explicit timestamp for the document
ttl	(aka "time to live") Expiration time for the document. Expired documents will be expunged automatically. The expiration date that will be set for a document with a provided ttl is relative to the timestamp of the document, meaning it can be based on the time of indexing or on any time provided. The provided ttl must be strictly positive and can be a number (in milliseconds) or any valid time value (e.g, 86400000, 1d).
refresh	Refresh the index after performing the operation. See http://bit.ly/2ezW9Zr for details
timeout	(character) Explicit operation timeout, e.g,. 5m (for 5 minutes)
retry_on_conflict	Specify how many times should the operation be retried when a conflict occurs (default: 0)
wait_for_active_shards	The number of shard copies required to be active before proceeding with the update operation. See http://bit.ly/2fbqkZ1 for details.
detect_noop	(logical) Specifying TRUE will cause Elasticsearch to check if there are changes and, if there aren't, turn the update request into a noop.
callopts	Curl options passed on to httr::POST()
...	Further args to query DSL

References

http://bit.ly/2eVYqLz

Examples

```
## Not run:
connect()
if (!index_exists('plos')) {
  plosdat <- system.file("examples", "plos_data.json", package = "elastic")
  invisible(docs_bulk(plosdat))
}

docs_create(index='plos', type='article', id=1002,
  body=list(id="12345", title="New title"))
# and the document is there now
docs_get(index='plos', type='article', id=1002)
# update the document
docs_update(index='plos', type='article', id=1002,
  body = list(doc = list(title = "Even newer title again")))
# get it again, notice changes
docs_get(index='plos', type='article', id=1002)

if (!index_exists('stuffthings')) {
  index_create("stuffthings")
}
docs_create(index='stuffthings', type='thing', id=1,
  body=list(name = "foo", what = "bar"))
docs_update(index='stuffthings', type='thing', id=1,
  body = list(doc = list(name = "hello", what = "bar")),
  source = 'name')

## End(Not run)
```

`documents` *Elasticsearch documents functions.*

Description

Elasticsearch documents functions.

Details

There are five functions to work directly with documents.

- `docs_get()`
- `docs_mget()`
- `docs_create()`
- `docs_delete()`
- `docs_bulk()`

Examples

```
## Not run:
# Get a document
# docs_get(index='plos', type='article', id=1)

# Get multiple documents
# docs_mget(index="shakespeare", type="line", id=c(9,10))

# Create a document
# docs_create(index='plos', type='article', id=35, body=list(id="12345", title="New title"))

# Delete a document
# docs_delete(index='plos', type='article', id=35)

# Bulk load documents
# plosdat <- system.file("examples", "plos_data.json", package = "elastic")
# docs_bulk(plosdat)

## End(Not run)
```

`elastic` *elastic: An Elasticsearch R client.*

Description

elastic: An Elasticsearch R client.

About

This package gives you access to local or remote Elasticsearch databases.

Quick start

If you're connecting to a Elasticsearch server already running, skip ahead to **Search**

Install Elasticsearch (on OSX)

- Download zip or tar file from Elasticsearch see here for download: https://www.elastic.co/downloads/elasticsearch
- Unzip it: `untar elasticsearch-2.3.5.tar.gz`
- Move it: `sudo mv elasticsearch-2.3.5 /usr/local` (replace version with your version)
- Navigate to /usr/local: `cd /usr/local`
- Add shortcut: `sudo ln -s elasticsearch-2.3.5 elasticsearch` (replace version with your version)

For help on other platforms, see https://www.elastic.co/guide/en/elasticsearch/reference/current/_installation.html

Start Elasticsearch

- Navigate to elasticsearch: `cd /usr/local/elasticsearch`
- Start elasticsearch: `bin/elasticsearch`

Initialization

The function `connect()` is used before doing anything else to set the connection details to your remote or local elasticsearch store. The details created by `connect()` are written to your options for the current session, and are used by `elastic` functions.

Search

The main way to search Elasticsearch is via the `Search()` function. E.g.:

```
Search()
```

Security

Elasticsearch is insecure out of the box! If you are running Elasticsearch locally on your own machine without exposing a port to the outside world, no worries, but if you install on a server with a public IP address, take the necessary precautions. There are a few options:

- Shield <https://www.elastic.co/products/shield‘ - This is a paid product - so probably only applicable to enterprise users
- DIY security - there are a variety of techniques for securing your Elasticsearch. I collected a number of resources in a blog post at http://recology.info/2015/02/secure-elasticsearch/

Elasticsearch changes

As of Elasticsearch v2:

- You can no longer create fields with dots in the name.
- Type names may not start with a dot (other than the special `.percolator` type)
- Type names may not be longer than 255 characters
- Types may no longer be deleted
- Queries and filters have been merged - all filter clauses are now query clauses. Instead, query clauses can now be used in query context or in filter context. See examples in `Search()` or `Search_uri()`

index names

The following are illegal characters, and can not be used in index names or types: \, /, *, ?, <, >, |, , (comma). double quote and whitespace are also illegal.

Author(s)

Scott Chamberlain <myrmecocystus@gmail.com>

elastic-defunct *Defunct functions in elastic*

Description

- `mlt()`: The MLT API has been removed, use More Like This Query via `Search()`
- `nodes_shutdown()`: The _shutdown API has been removed. Instead, setup Elasticsearch to run as a service (see Running as a Service on Linux (https://www.elastic.co/guide/en/elasticsearch/reference/2.0/setup-service.html) or Running as a Service on Windows (https://www.elastic.co/guide/en/elasticsearch/reference/2.0/setup-service-win.html)) or use the -p command line option to write the PID to a file.
- `index_status()`: _status route for the index API has been removed. Replaced with the Indices Stats and Indices Recovery APIs.
- `mapping_delete()`: Elasticsearch dropped this route in their API. Instead of deleting a mapping, delete the index and recreate with a new mapping.

explain *Explain a search query.*

Description

Explain a search query.

Usage

```
explain(index = NULL, type = NULL, id = NULL, source2 = NULL,
  fields = NULL, routing = NULL, parent = NULL, preference = NULL,
  source = NULL, q = NULL, df = NULL, analyzer = NULL,
  analyze_wildcard = NULL, lowercase_expanded_terms = NULL,
  lenient = NULL, default_operator = NULL, source_exclude = NULL,
  source_include = NULL, body = NULL, raw = FALSE, ...)
```

Arguments

index	Only one index
type	Only one document type
id	Document id, only one
source2	(logical) Set to TRUE to retrieve the _source of the document explained. You can also retrieve part of the document by using source_include & source_exclude (see Get API for more details). This matches the `_source` term, but we want to avoid the leading underscore.
fields	Allows to control which stored fields to return as part of the document explained.
routing	Controls the routing in the case the routing was used during indexing.

parent	Same effect as setting the routing parameter.
preference	Controls on which shard the explain is executed.
source	Allows the data of the request to be put in the query string of the url.
q	The query string (maps to the query_string query).
df	The default field to use when no field prefix is defined within the query. Defaults to _all field.
analyzer	The analyzer name to be used when analyzing the query string. Defaults to the analyzer of the _all field.
analyze_wildcard	(logical) Should wildcard and prefix queries be analyzed or not. Default: FALSE
lowercase_expanded_terms	Should terms be automatically lowercased or not. Default: TRUE
lenient	If set to true will cause format based failures (like providing text to a numeric field) to be ignored. Default: FALSE
default_operator	The default operator to be used, can be AND or OR. Defaults to OR.
source_exclude	A vector of fields to exclude from the returned source2 field
source_include	A vector of fields to extract and return from the source2 field
body	The query definition using the Query DSL. This is passed in the body of the request.
raw	If TRUE (default), data is parsed to list. If FALSE, then raw JSON.
...	Curl args passed on to httr::GET()

References

https://www.elastic.co/guide/en/elasticsearch/reference/current/search-explain.html

Examples

```
## Not run:
explain(index = "plos", type = "article", id = 14, q = "title:Germ")

body <- '{
 "query": {
   "term": { "title": "Germ" }
 }
}'
explain(index = "plos", type = "article", id = 14, body=body)

## End(Not run)
```

fielddata *fielddata*

Description

Deep dive on fielddata details

Details

Most fields are indexed by default, which makes them searchable. Sorting, aggregations, and accessing field values in scripts, however, requires a different access pattern from search.

Text fields use a query-time in-memory data structure called fielddata. This data structure is built on demand the first time that a field is used for aggregations, sorting, or in a script. It is built by reading the entire inverted index for each segment from disk, inverting the term-document relationship, and storing the result in memory, in the JVM heap.

fielddata is disabled on text fields by default. Fielddata can consume a lot of heap space, especially when loading high cardinality text fields. Once fielddata has been loaded into the heap, it remains there for the lifetime of the segment. Also, loading fielddata is an expensive process which can cause users to experience latency hits. This is why fielddata is disabled by default. If you try to sort, aggregate, or access values from a script on a text field, you will see this exception:

"Fielddata is disabled on text fields by default. Set fielddata=true on `your_field_name` in order to load fielddata in memory by uninverting the inverted index. Note that this can however use significant memory."

To enable fielddata on a text field use the PUT mapping API, for example `mapping_create("shakespeare", body = '{ "properties": { "speaker": { "type": "text", "fielddata": true } } }')`

You may get an error about `update_all_types`, in which case set `update_all_types=TRUE` in `mapping_create`, e.g.,

```
mapping_create("shakespeare", update_all_types=TRUE, body = '{ "properties": { "speaker": { "type": "text", "fielddata": true } } }')
```

See https://www.elastic.co/guide/en/elasticsearch/reference/current/fielddata.html#_enabling_fielddata_on_literal_text_literal_fields for more information.

field_caps *Field capabilities*

Description

The field capabilities API allows to retrieve the capabilities of fields among multiple indices.

Usage

```
field_caps(fields = NULL, index = NULL, body = list(), raw = FALSE,
  asdf = FALSE, ...)
```

Arguments

fields	A list of fields to compute stats for. optional
index	Index name, one or more
body	Query, either a list or json
raw	(logical) Get raw JSON back or not
asdf	(logical) If TRUE, use jsonlite::fromJSON() to parse JSON directly to a data.frame if possible. If FALSE (default), list output is given.
...	Curl args passed on to httr::POST()

References

https://www.elastic.co/guide/en/elasticsearch/reference/current/search-field-caps.html

See Also

field_stats()

Examples

```
## Not run:
connect()

field_caps(body = '{ "fields": ["speaker"] }', index = "shakespeare")

## End(Not run)
```

---

field_stats *Search field statistics*

Description

Search field statistics

Usage

```
field_stats(fields = NULL, index = NULL, level = "cluster",
  body = list(), raw = FALSE, asdf = FALSE, ...)
```

Arguments

fields	A list of fields to compute stats for. optional
index	Index name, one or more
level	Defines if field stats should be returned on a per index level or on a cluster wide level. Valid values are 'indices' and 'cluster' (default)

body	Query, either a list or json
raw	(logical) Get raw JSON back or not
asdf	(logical) If TRUE, use fromJSON to parse JSON directly to a data.frame. If FALSE (Default), list output is given.
...	Curl args passed on to httr::POST()

Details

The field stats api allows you to get statistical properties of a field without executing a search, but looking up measurements that are natively available in the Lucene index. This can be useful to explore a dataset which you don't know much about. For example, this allows creating a histogram aggregation with meaningful intervals based on the min/max range of values.

The field stats api by defaults executes on all indices, but can execute on specific indices too.

Note

Deprecated in Elasticsearch versions equal to/greater than 5.4.0

References

https://www.elastic.co/guide/en/elasticsearch/reference/5.6/search-field-stats.html

See Also

field_caps()

Examples

```
## Not run:
connect()

if (gsub("\\.", "", ping()$version$number) < 500) {
  field_stats(body = '{ "fields": ["speaker"] }', index = "shakespeare")
  ff <- c("scientificName", "continent", "decimalLatitude", "play_name",
    "speech_number")
  field_stats("play_name")
  field_stats("play_name", level = "cluster")
  field_stats(ff, level = "indices")
  field_stats(ff)
  field_stats(ff, index = c("gbif", "shakespeare"))

  # can also pass a body, just as with Search()
  # field_stats(body = list(fields = "rating")) # doesn't work
  field_stats(body = '{ "fields": ["scientificName"] }', index = "gbif")

  body <- '{
    "fields" : ["scientificName", "decimalLatitude"]
  }'
  field_stats(body = body, level = "indices", index = "gbif")
```

```
}

## End(Not run)
```

---

index *Index API operations*

Description

Index API operations

Usage

```
index_get(index = NULL, features = NULL, raw = FALSE, verbose = TRUE,
  ...)

index_exists(index, ...)

index_delete(index, raw = FALSE, verbose = TRUE, ...)

index_create(index = NULL, body = NULL, raw = FALSE, verbose = TRUE,
  ...)

index_recreate(index = NULL, body = NULL, raw = FALSE, verbose = TRUE,
  ...)

index_close(index, ...)

index_open(index, ...)

index_stats(index = NULL, metric = NULL, completion_fields = NULL,
  fielddata_fields = NULL, fields = NULL, groups = NULL,
  level = "indices", ...)

index_settings(index = "_all", ...)

index_settings_update(index = NULL, body, ...)

index_segments(index = NULL, ...)

index_recovery(index = NULL, detailed = FALSE, active_only = FALSE, ...)

index_optimize(index = NULL, max_num_segments = NULL,
  only_expunge_deletes = FALSE, flush = TRUE, wait_for_merge = TRUE, ...)

index_forcemerge(index = NULL, max_num_segments = NULL,
  only_expunge_deletes = FALSE, flush = TRUE, ...)
```

```
index_upgrade(index = NULL, wait_for_completion = FALSE, ...)

index_analyze(text = NULL, field = NULL, index = NULL, analyzer = NULL,
  tokenizer = NULL, filters = NULL, char_filters = NULL, body = list(),
  ...)

index_flush(index = NULL, force = FALSE, full = FALSE,
  wait_if_ongoing = FALSE, ...)

index_clear_cache(index = NULL, filter = FALSE, filter_keys = NULL,
  fielddata = FALSE, query_cache = FALSE, id_cache = FALSE, ...)
```

Arguments

index	(character) A character vector of index names
features	(character) A single feature. One of settings, mappings, or aliases
raw	If TRUE (default), data is parsed to list. If FALSE, then raw JSON.
verbose	If TRUE (default) the url call used printed to console.
...	Curl args passed on to `httr::POST()`, `httr::GET()`, `httr::PUT()`, `httr::HEAD()`, or `httr::DELETE()`
body	Query, either a list or json.
metric	(character) A character vector of metrics to display. Possible values: "_all", "completion", "docs", "fielddata", "filter_cache", "flush", "get", "id_cache", "indexing", "merge", "percolate", "refresh", "search", "segments", "store", "warmer".
completion_fields	(character) A character vector of fields for completion metric (supports wildcards)
fielddata_fields	(character) A character vector of fields for fielddata metric (supports wildcards)
fields	(character) Fields to add.
groups	(character) A character vector of search groups for search statistics.
level	(character) Return stats aggregated on "cluster", "indices" (default) or "shards"
detailed	(logical) Whether to display detailed information about shard recovery. Default: FALSE
active_only	(logical) Display only those recoveries that are currently on-going. Default: FALSE
max_num_segments	(character) The number of segments the index should be merged into. Default: "dynamic"
only_expunge_deletes	(logical) Specify whether the operation should only expunge deleted documents
flush	(logical) Specify whether the index should be flushed after performing the operation. Default: TRUE

wait_for_merge (logical) Specify whether the request should block until the merge process is finished. Default: TRUE

wait_for_completion
(logical) Should the request wait for the upgrade to complete. Default: FALSE

text The text on which the analysis should be performed (when request body is not used)

field Use the analyzer configured for this field (instead of passing the analyzer name)

analyzer The name of the analyzer to use

tokenizer The name of the tokenizer to use for the analysis

filters A character vector of filters to use for the analysis

char_filters A character vector of character filters to use for the analysis

force (logical) Whether a flush should be forced even if it is not necessarily needed ie. if no changes will be committed to the index.

full (logical) If set to TRUE a new index writer is created and settings that have been changed related to the index writer will be refreshed.

wait_if_ongoing
If TRUE, the flush operation will block until the flush can be executed if another flush operation is already executing. The default is false and will cause an exception to be thrown on the shard level if another flush operation is already running.

filter (logical) Clear filter caches

filter_keys (character) A vector of keys to clear when using the filter_cache parameter (default: all)

fielddata (logical) Clear field data

query_cache (logical) Clear query caches

id_cache (logical) Clear ID caches for parent/child

Details

index_analyze: https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-analyze.html This method can accept a string of text in the body, but this function passes it as a parameter in a GET request to simplify.

index_flush: https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-flush.html From the ES website: The flush process of an index basically frees memory from the index by flushing data to the index storage and clearing the internal transaction log. By default, Elasticsearch uses memory heuristics in order to automatically trigger flush operations as required in order to clear memory.

index_status: The API endpoint for this function was deprecated in Elasticsearch v1.2.0, and will likely be removed soon. Use index_recovery() instead.

index_settings_update: There are a lot of options you can change with this function. See https://www.elastic.co/guide/en/elas
update-settings.html for all the options.

Author(s)

Scott Chamberlain myrmecocystus@gmail.com

References

https://www.elastic.co/guide/en/elasticsearch/reference/current/indices.html

Examples

```
## Not run:
# get information on an index
index_get(index='shakespeare')
## this one is the same as running index_settings('shakespeare')
index_get(index='shakespeare', features='settings')
index_get(index='shakespeare', features='mappings')
index_get(index='shakespeare', features='alias')

# check for index existence
index_exists(index='shakespeare')
index_exists(index='plos')

# create an index
if (index_exists('twitter')) index_delete('twitter')
index_create(index='twitter')
if (index_exists('things')) index_delete('things')
index_create(index='things')
if (index_exists('plos')) index_delete('plos')
index_create(index='plos')

# re-create an index
index_recreate("deer")
index_recreate("deer", verbose = FALSE)

# delete an index
if (index_exists('plos')) index_delete(index='plos')

## with a body
body <- '{
 "settings" : {
  "index" : {
    "number_of_shards" : 3,
    "number_of_replicas" : 2
   }
 }
}'
if (index_exists('alsothat')) index_delete('alsothat')
index_create(index='alsothat', body=body)

## with mappings
body <- '{
 "mappings": {
   "record": {
     "properties": {
       "location" : {"type" : "geo_point"}
      }
   }
```

```
 }
}'
if (!index_exists('gbifnewgeo')) index_create(index='gbifnewgeo', body=body)
gbifgeo <- system.file("examples", "gbif_geosmall.json", package = "elastic")
docs_bulk(gbifgeo)

# close an index
index_create('plos')
index_close('plos')

# open an index
index_open('plos')

# Get stats on an index
index_stats('plos')
index_stats(c('plos','gbif'))
index_stats(c('plos','gbif'), metric='refresh')
index_stats(metric = "indexing")
index_stats('shakespeare', metric='completion')
index_stats('shakespeare', metric='completion', completion_fields = "completion")
index_stats('shakespeare', metric='fielddata')
index_stats('shakespeare', metric='fielddata', fielddata_fields = "evictions")
index_stats('plos', level="indices")
index_stats('plos', level="cluster")
index_stats('plos', level="shards")

# Get segments information that a Lucene index (shard level) is built with
index_segments()
index_segments('plos')
index_segments(c('plos','gbif'))

# Get recovery information that provides insight into on-going index shard recoveries
index_recovery()
index_recovery('plos')
index_recovery(c('plos','gbif'))
index_recovery("plos", detailed = TRUE)
index_recovery("plos", active_only = TRUE)

# Optimize an index, or many indices
if (gsub("\\.", "", ping()$version$number) < 500) {
  ### ES < v5 - use optimize
  index_optimize('plos')
  index_optimize(c('plos','gbif'))
  index_optimize('plos')
} else {
  ### ES > v5 - use forcemerge
  index_forcemerge('plos')
}

# Upgrade one or more indices to the latest format. The upgrade process converts any
# segments written with previous formats.
if (gsub("\\.", "", ping()$version$number) < 500) {
  index_upgrade('plos')
```

```
  index_upgrade(c('plos','gbif'))
}

# Performs the analysis process on a text and return the tokens breakdown
# of the text
index_analyze(text = 'this is a test', analyzer='standard')
index_analyze(text = 'this is a test', analyzer='whitespace')
index_analyze(text = 'this is a test', analyzer='stop')
index_analyze(text = 'this is a test', tokenizer='keyword',
  filters='lowercase')
index_analyze(text = 'this is a test', tokenizer='keyword', filters='lowercase',
   char_filters='html_strip')
index_analyze(text = 'this is a test', index = 'plos')
index_analyze(text = 'this is a test', index = 'shakespeare')
index_analyze(text = 'this is a test', index = 'shakespeare',
  config=verbose())

## NGram tokenizer
body <- '{
        "settings" : {
             "analysis" : {
                 "analyzer" : {
                     "my_ngram_analyzer" : {
                         "tokenizer" : "my_ngram_tokenizer"
                     }
                 },
                 "tokenizer" : {
                     "my_ngram_tokenizer" : {
                         "type" : "nGram",
                         "min_gram" : "2",
                         "max_gram" : "3",
                         "token_chars": [ "letter", "digit" ]
                     }
                 }
             }
      }
}'
if (index_exists("shakespeare2")) {
   index_delete("shakespeare2")
}
tokenizer_set(index = "shakespeare2", body=body)
index_analyze(text = "art thouh", index = "shakespeare2",
  analyzer='my_ngram_analyzer')

# Explicitly flush one or more indices.
index_flush(index = "plos")
index_flush(index = "shakespeare")
index_flush(index = c("plos","shakespeare"))
index_flush(index = "plos", wait_if_ongoing = TRUE)
library('httr')
index_flush(index = "plos", config=verbose())

# Clear either all caches or specific cached associated with one ore more indices.
```

```
index_clear_cache()
index_clear_cache(index = "plos")
index_clear_cache(index = "shakespeare")
index_clear_cache(index = c("plos","shakespeare"))
index_clear_cache(filter = TRUE)
library('httr')
index_clear_cache(config=verbose())

# Index settings
## get settings
index_settings()
index_settings("_all")
index_settings('gbif')
index_settings(c('gbif','plos'))
index_settings('*s')
## update settings
if (index_exists('foobar')) index_delete('foobar')
index_create("foobar")
settings <- list(index = list(number_of_replicas = 4))
index_settings_update("foobar", body = settings)
index_get("foobar")$foobar$settings

## End(Not run)
```

`index_template` *Index templates*

Description

Index templates allow you to define templates that will automatically be applied when new indices are created

Usage

```
index_template_put(name, body = NULL, create = NULL, flat_settings = NULL,
  master_timeout = NULL, order = NULL, timeout = NULL, ...)

index_template_get(name = NULL, filter_path = NULL, ...)

index_template_exists(name, ...)

index_template_delete(name, ...)
```

Arguments

`name`	(character) The name of the template
`body`	(character/list) The template definition
`create`	(logical) Whether the index template should only be added if new or can also replace an existing one. Default: `FALSE`

flat_settings	(logical) Return settings in flat format. Default: FALSE
master_timeout	(integer) Specify timeout for connection to master
order	(integer) The order for this template when merging multiple matching ones (higher numbers are merged later, overriding the lower numbers)
timeout	(integer) Explicit operation timeout
...	Curl options. Or in percolate_list function, further args passed on to Search()
filter_path	(character) a regex for filtering output path, see example

References

https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-templates.html

Examples

```
## Not run:
body <- '{
  "template": "te*",
  "settings": {
    "number_of_shards": 1
  },
  "mappings": {
    "type1": {
      "_source": {
        "enabled": false
      },
      "properties": {
        "host_name": {
          "type": "keyword"
        },
        "created_at": {
          "type": "date",
          "format": "EEE MMM dd HH:mm:ss Z YYYY"
        }
      }
    }
  }
}'
index_template_put("template_1", body = body)

# get templates
index_template_get()
index_template_get("template_1")
index_template_get(c("template_1", "template_2"))
index_template_get("template_*")
## filter path
index_template_get("template_1", filter_path = "*.template")

# template exists
index_template_exists("template_1")
```

```
index_template_exists("foobar")

# delete a template
index_template_delete("template_1")
index_template_exists("template_1")

## End(Not run)
```

`info` *Get the basic info from the current cluster*

Description

Get the basic info from the current cluster

Usage

```
info(...)
```

Arguments

`...`	Further args passed on to print for the es_conn class.

Examples

```
## Not run:
connect()
info()

## End(Not run)
```

`mapping` *Mapping management*

Description

Mapping management

Usage

```
mapping_create(index, type, body, update_all_types = FALSE, ...)

mapping_get(index = NULL, type = NULL, ...)

field_mapping_get(index = NULL, type = NULL, field,
  include_defaults = FALSE, ...)

type_exists(index, type, ...)
```

Arguments

index	(character) An index
type	(character) A document type
body	(list) Either a list or json, representing the query.
update_all_types	(logical) update all types. default: FALSE. This parameter is deprecated in ES v6.3.0 and higher, see https://github.com/elastic/elasticsearch/pull/28284
...	Curl options passed on to httr::HEAD() or other http verbs
field	(character) One or more field names
include_defaults	(logical) Whether to return default values

Details

Find documentation for each function at:

- mapping_create - https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-put-mapping.html
- type_exists - https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-types-exists.html
- mapping_delete - FUNCTION DEFUNCT - instead of deleting mapping, delete index and recreate index with new mapping
- mapping_get - https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-get-mapping.html
- field_mapping_get - https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-get-field-mapping.html

Examples

```
## Not run:
# Used to check if a type/types exists in an index/indices
type_exists(index = "plos", type = "article")
type_exists(index = "plos", type = "articles")
type_exists(index = "shakespeare", type = "line")

# The put mapping API allows to register specific mapping definition for a specific type.
## a good mapping body
body <- list(citation = list(properties = list(
 journal = list(type="text"),
 year = list(type="long")
)))
if (!index_exists("plos")) index_create("plos")
mapping_create(index = "plos", type = "citation", body=body)

### or as json
body <- '{
  "citation": {
```

```
    "properties": {
      "journal": { "type": "text" },
      "year": { "type": "long" }
}}}'
mapping_create(index = "plos", type = "citation", body=body)
mapping_get("plos", "citation")

## A bad mapping body
body <- list(things = list(properties = list(
  journal = list("text")
)))
# mapping_create(index = "plos", type = "things", body=body)

# Get mappings
mapping_get('_all')
mapping_get(index = "plos")
mapping_get(index = c("shakespeare","plos"))
mapping_get(index = "shakespeare", type = "act")
mapping_get(index = "shakespeare", type = c("act","line"))

# Get field mappings
plosdat <- system.file("examples", "plos_data.json", package = "elastic")
invisible(docs_bulk(plosdat))
field_mapping_get(index = "_all", type=c('article', 'line'), field = "text")
field_mapping_get(index = "plos", type = "article", field = "title")
field_mapping_get(index = "plos", type = "article", field = "*")
field_mapping_get(index = "plos", type = "article", field = "title", include_defaults = TRUE)
field_mapping_get(type = c("article","record"), field = c("title","class"))
field_mapping_get(type = "a*", field = "t*")

# Create geospatial mapping
if (index_exists("gbifgeopoint")) index_delete("gbifgeopoint")
file <- system.file("examples", "gbif_geopoint.json", package = "elastic")
index_create("gbifgeopoint")
body <- '{
 "properties" : {
   "location" : { "type" : "geo_point" }
 }
}'
mapping_create("gbifgeopoint", "record", body = body)
invisible(docs_bulk(file))

# update_all_fields, see also ?fielddata
if (es_ver() < 603) {
 mapping_create("shakespeare", "record", update_all_types=TRUE, body = '{
   "properties": {
     "speaker": {
       "type":     "text",
       "fielddata": true
     }
   }
 }')
} else {
```

```
 index_create('brownchair')
 mapping_create('brownchair', 'brown', body = '{
   "properties": {
     "foo": {
       "type":     "text",
       "fielddata": true
     }
   }
 }')
}


## End(Not run)
```

`msearch` *Multi-search*

Description

Performs multiple searches, defined in a file

Usage

```
msearch(x, raw = FALSE, asdf = FALSE, ...)
```

Arguments

`x`	(character) A file path
`raw`	(logical) Get raw JSON back or not.
`asdf`	(logical) If `TRUE`, use `jsonlite::fromJSON()` to parse JSON directly to a data.frame. If `FALSE` (Default), list output is given.
`...`	Curl args passed on to `httr::POST()`

Details

This function behaves similarly to `docs_bulk()` - performs searches based on queries defined in a file.

See Also

`Search_uri()` `Search()`

Examples

```
## Not run:
connect()
msearch1 <- system.file("examples", "msearch_eg1.json", package = "elastic")
readLines(msearch1)
msearch(msearch1)

cat('{"index" : "shakespeare"}', file = "~/mysearch.json", sep = "\n")
cat('{"query" : {"match_all" : {}}, "from" : 0, "size" : 5}',  sep = "\n",
   file = "~/mysearch.json", append = TRUE)
msearch("~/mysearch.json")

## End(Not run)
```

`mtermvectors` *Multi Termvectors*

Description

Multi Termvectors

Usage

```
mtermvectors(index = NULL, type = NULL, ids = NULL, body = list(),
  pretty = TRUE, field_statistics = TRUE, fields = NULL, offsets = TRUE,
  parent = NULL, payloads = TRUE, positions = TRUE,
  preference = "random", realtime = TRUE, routing = NULL,
  term_statistics = FALSE, version = NULL, version_type = NULL, ...)
```

Arguments

`index`	(character) The index in which the document resides.
`type`	(character) The type of the document.
`ids`	(character) One or more document ids
`body`	(character) Define parameters and or supply a document to get termvectors for
`pretty`	(logical) pretty print. Default: `TRUE`
`field_statistics`	(character) Specifies if document count, sum of document frequencies and sum of total term frequencies should be returned. Default: `TRUE`
`fields`	(character) A comma-separated list of fields to return.
`offsets`	(character) Specifies if term offsets should be returned. Default: `TRUE`
`parent`	(character) Parent id of documents.
`payloads`	(character) Specifies if term payloads should be returned. Default: `TRUE`
`positions`	(character) Specifies if term positions should be returned. Default: `TRUE`

preference	(character) Specify the node or shard the operation should be performed on (Default: random).
realtime	(character) Specifies if request is real-time as opposed to near-real-time (Default: TRUE).
routing	(character) Specific routing value.
term_statistics	(character) Specifies if total term frequency and document frequency should be returned. Default: FALSE
version	(character) Explicit version number for concurrency control
version_type	(character) Specific version type, valid choices are: 'internal', 'external', 'external_gte', 'force'
...	Curl args passed on to httr::POST()

Details

Multi termvectors API allows to get multiple termvectors based on an index, type and id.

References

https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-multi-termvectors.html

Examples

```
## Not run:
connect()
if (!index_exists('omdb')) {
  omdb <- system.file("examples", "omdb.json", package = "elastic")
  docs_bulk(omdb)
}

# no index or type given
body <- '{
   "docs": [
      {
         "_index": "omdb",
         "_type": "omdb",
         "_id": "AVXdx8Eqg_0Z_tpMDyP_",
         "term_statistics": true
      },
      {
         "_index": "omdb",
         "_type": "omdb",
         "_id": "AVXdx8Eqg_0Z_tpMDyQ1",
         "fields": [
            "Plot"
         ]
      }
   ]
}'
```

```
mtermvectors(body = body)

# index given, but not type
body <- '{
   "docs": [
      {
         "_type": "omdb",
         "_id": "AVXdx8Eqg_0Z_tpMDyP_",
         "fields": [
            "Plot"
         ],
         "term_statistics": true
      },
      {
         "_type": "omdb",
         "_id": "AVXdx8Eqg_0Z_tpMDyQ1",
         "fields": [
            "Title"
         ]
      }
   ]
}'
mtermvectors('omdb', body = body)

# index and type given
body <- '{
   "docs": [
      {
         "_id": "AVXdx8Eqg_0Z_tpMDyP_",
         "fields": [
            "Plot"
         ],
         "term_statistics": true
      },
      {
         "_id": "AVXdx8Eqg_0Z_tpMDyQ1"
      }
   ]
}'
mtermvectors('omdb', 'omdb', body = body)

# index and type given, parameters same, so can simplify
body <- '{
    "ids" : ["AVXdx8Eqg_0Z_tpMDyP_", "AVXdx8Eqg_0Z_tpMDyQ1"],
    "parameters": {
        "fields": [
            "Plot"
        ],
        "term_statistics": true
    }
}'
mtermvectors('omdb', 'omdb', body = body)
```

```
# you can give user provided documents via the 'docs' parameter
## though you have to give index and type that exist in your Elasticsearch
## instance
body <- '{
   "docs": [
      {
         "_index": "omdb",
         "_type": "omdb",
         "doc" : {
            "Director" : "John Doe",
            "Plot" : "twitter test test test"
         }
      },
      {
         "_index": "omdb",
         "_type": "omdb",
         "doc" : {
           "Director" : "Jane Doe",
           "Plot" : "Another twitter test ..."
         }
      }
   ]
}'
mtermvectors(body = body)

## End(Not run)
```

---

nodes *Elasticsearch nodes endpoints.*

Description

Elasticsearch nodes endpoints.

Usage

```
nodes_stats(node = NULL, metric = NULL, raw = FALSE, fields = NULL,
  verbose = TRUE, ...)

nodes_info(node = NULL, metric = NULL, raw = FALSE, verbose = TRUE, ...)

nodes_hot_threads(node = NULL, metric = NULL, threads = 3,
  interval = "500ms", type = NULL, raw = FALSE, verbose = TRUE, ...)
```

Arguments

node	The node
metric	A metric to get. See Details.
raw	If TRUE (default), data is parsed to list. If FALSE, then raw JSON.

fields	You can get information about field data memory usage on node level or on index level
verbose	If TRUE (default) the url call used printed to console
...	Curl args passed on to GET
threads	(character) Number of hot threads to provide. Default: 3
interval	(character) The interval to do the second sampling of threads. Default: 500ms
type	(character) The type to sample, defaults to cpu, but supports wait and block to see hot threads that are in wait or block state.

Details

https://www.elastic.co/guide/en/elasticsearch/reference/current/cluster-nodes-stats.html

By default, all stats are returned. You can limit this by combining any of indices, os, process, jvm, network, transport, http, fs, breaker and thread_pool. With the metric parameter you can select zero or more of:

- indices Indices stats about size, document count, indexing and deletion times, search times, field cache size, merges and flushes
- os retrieve information that concern the operating system
- fs File system information, data path, free disk space, read/write stats
- http HTTP connection information
- jvm JVM stats, memory pool information, garbage collection, buffer pools
- network TCP information
- os Operating system stats, load average, cpu, mem, swap
- process Process statistics, memory consumption, cpu usage, open file descriptors
- thread_pool Statistics about each thread pool, including current size, queue and rejected tasks
- transport Transport statistics about sent and received bytes in cluster communication
- breaker Statistics about the field data circuit breaker

nodes_hot_threads() returns plain text, so base::cat() is used to print to the console.

Examples

```
## Not run:
(out <- nodes_stats())
nodes_stats(node = names(out$nodes))
nodes_stats(metric='get')
nodes_stats(metric='jvm')
nodes_stats(metric=c('os','process'))
nodes_info()
nodes_info(metric='process')
nodes_info(metric='jvm')
nodes_info(metric='http')
nodes_info(metric='network')

## End(Not run)
```

percolate *Percolater*

Description

Store queries into an index then, via the percolate API, define documents to retrieve these queries.

Usage

```
percolate_register(index, type = NULL, id, body = list(), routing = NULL,
  preference = NULL, ignore_unavailable = NULL, percolate_format = NULL,
  refresh = NULL, ...)

percolate_match(index, type = NULL, body, routing = NULL,
  preference = NULL, ignore_unavailable = NULL, percolate_format = NULL,
  ...)

percolate_list(index, ...)

percolate_count(index, type, body, ...)

percolate_delete(index, id)
```

Arguments

index	Index name. Required
type	Document type
id	A precolator id. Required
body	Body json, or R list.
routing	(character) In case the percolate queries are partitioned by a custom routing value, that routing option makes sure that the percolate request only gets executed on the shard where the routing value is partitioned to. This means that the percolate request only gets executed on one shard instead of all shards. Multiple values can be specified as a comma separated string, in that case the request can be be executed on more than one shard.
preference	(character) Controls which shard replicas are preferred to execute the request on. Works the same as in the search API.
ignore_unavailable	(logical) Controls if missing concrete indices should silently be ignored. Same as is in the search API.
percolate_format	(character) If ids is specified then the matches array in the percolate response will contain a string array of the matching ids instead of an array of objects. This can be useful to reduce the amount of data being send back to the client. Obviously if there are two percolator queries with same id from different indices there is no way to find out which percolator query belongs to what index. Any other value to percolate_format will be ignored.

refresh	If TRUE then refresh the affected shards to make this operation visible to search, if "wait_for" then wait for a refresh to make this operation visible to search, if FALSE (default) then do nothing with refreshes. Valid choices: TRUE, FALSE, "wait_for"
...	Curl options. Or in percolate_list function, further args passed on to Search()

Details

Additional body options, pass those in the body. These aren't query string parameters:

- filter - Reduces the number queries to execute during percolating. Only the percolator queries that match with the filter will be included in the percolate execution. The filter option works in near realtime, so a refresh needs to have occurred for the filter to included the latest percolate queries.
- query - Same as the filter option, but also the score is computed. The computed scores can then be used by the track_scores and sort option.
- size - Defines to maximum number of matches (percolate queries) to be returned. Defaults to unlimited.
- track_scores - Whether the _score is included for each match. The _score is based on the query and represents how the query matched the percolate query's metadata, not how the document (that is being percolated) matched the query. The query option is required for this option. Defaults to false.
- sort - Define a sort specification like in the search API. Currently only sorting _score reverse (default relevancy) is supported. Other sort fields will throw an exception. The size and query option are required for this setting. Like track_score the score is based on the query and represents how the query matched to the percolate query's metadata and not how the document being percolated matched to the query.
- aggs - Allows aggregation definitions to be included. The aggregations are based on the matching percolator queries, look at the aggregation documentation on how to define aggregations.
- highlight - Allows highlight definitions to be included. The document being percolated is being highlight for each matching query. This allows you to see how each match is highlighting the document being percolated. See highlight documentation on how to define highlights. The size option is required for highlighting, the performance of highlighting in the percolate API depends of how many matches are being highlighted.

The Elasticsearch v5 split

In Elasticsearch < v5, there's a certain set of percolate APIs available, while in Elasticsearch >= v5, there's a different set of APIs available.

Internally within these percolate functions we detect your Elasticsearch version, then use the appropriate APIs

References

https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-percolate-query.html

Examples

```
## Not run:
##### Elasticsearch < v5
# typical usage
## create an index first
if (index_exists("myindex")) index_delete("myindex")
mapping <- '{
  "mappings": {
    "mytype": {
      "properties": {
        "message": {
           "type": "text"
        },
        "name": {
           "type": "text"
        }
      }
    }
  }
}'
index_create("myindex", body = mapping)

## register a percolator
perc_body = '{
 "query" : {
    "match" : {
      "message" : "bonsai tree"
    }
 }
}'
percolate_register(index = "myindex", id = 1, body = perc_body)

## register another
perc_body2 <- '{
  "query" : {
    "match" : {
      "name" : "jane doe"
    }
  }
}'
percolate_register(index = "myindex", id = 2, body = perc_body2)

## match a document to a percolator
doc <- '{
  "doc" : {
    "message" : "A new bonsai tree in the office"
  }
}'
percolate_match(index = "myindex", type = "mytype", body = doc, config = verbose())

## List percolators - for an index, no type, can't do across indices
percolate_list(index = "myindex")$hits$hits
```

```
## Percolate counter
percolate_count(index = "myindex", type = "mytype", body = doc)$total

## delete a percolator
percolate_delete(index = "myindex", id = 2)

# multi percolate
## not working yet



##### Elasticsearch >= v5
if (index_exists("myindex")) index_delete("myindex")
body <- '{
  "mappings": {
    "doctype": {
      "properties": {
        "message": {
          "type": "text"
        }
      }
    },
    "queries": {
      "properties": {
        "query": {
          "type": "percolator"
        }
      }
    }
  }
}'

# create the index with mapping
index_create("myindex", body = body)

## register a percolator
x <- '{
  "query" : {
     "match" : {
       "message" : "bonsai tree"
     }
  }
}'
percolate_register(index = "myindex", type = "queries", id = 1, body = x)

## register another
x2 <- '{
  "query" : {
    "match" : {
      "message" : "the office"
    }
  }
```

```
}'
percolate_register(index = "myindex", type = "queries", id = 2, body = x2)

## match a document to a percolator
query <- '{
  "query" : {
    "percolate" : {
      "field": "query",
      "document_type": "doctype",
      "document": {
        "message": "A new bonsai tree in the office"
      }
    }
  }
}'
percolate_match(index = "myindex", body = query)

## End(Not run)
```

`ping` *Ping an Elasticsearch server.*

Description

Ping an Elasticsearch server.

Usage

```
ping(...)
```

Arguments

`...`	Curl args passed on to `httr::GET()`

See Also

`connect()`

Examples

```
## Not run:
ping()

## End(Not run)
```

`preference` *Preferences.*

Description

Preferences.

Details

- _primary The operation will go and be executed only on the primary shards.
- _primary_first The operation will go and be executed on the primary shard, and if not available (failover), will execute on other shards.
- _local The operation will prefer to be executed on a local allocated shard if possible.
- _only_node:xyz Restricts the search to execute only on a node with the provided node id (xyz in this case).
- _prefer_node:xyz Prefers execution on the node with the provided node id (xyz in this case) if applicable.
- _shards:2,3 Restricts the operation to the specified shards. (2 and 3 in this case). This preference can be combined with other preferences but it has to appear first: _shards:2,3;_primary
- Custom (string) value A custom value will be used to guarantee that the same shards will be used for the same custom value. This can help with "jumping values" when hitting different shards in different refresh states. A sample value can be something like the web session id, or the user name.

`reindex` *Reindex*

Description

Reindex all documents from one index to another.

Usage

```
reindex(body, refresh = NULL, requests_per_second = NULL, slices = NULL,
  timeout = NULL, wait_for_active_shards = NULL,
  wait_for_completion = NULL, ...)
```

Arguments

body	(list/character/json) The search definition using the Query DSL and the prototype for the index request.
refresh	(logical) Should the effected indexes be refreshed?
requests_per_second	(integer) The throttle to set on this request in sub-requests per second. - 1 means no throttle. Default: 0
slices	(integer) The number of slices this task should be divided into. Defaults to 1 meaning the task isn't sliced into subtasks. Default: 1
timeout	(character) Time each individual bulk request should wait for shards that are unavailable. Default: '1m'
wait_for_active_shards	(integer) Sets the number of shard copies that must be active before proceeding with the reindex operation. Defaults to 1, meaning the primary shard only. Set to all for all shard copies, otherwise set to any non-negative value less than or equal to the total number of copies for the shard (number of replicas + 1)
wait_for_completion	(logical) Should the request block until the reindex is complete? Default: TRUE
...	Curl options, passed on to httr::POST()

References

https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-reindex.html

Examples

```
## Not run:
if (!index_exists("twitter")) index_create("twitter")
if (!index_exists("new_twitter")) index_create("new_twitter")
body <- '{
  "source": {
    "index": "twitter"
  },
  "dest": {
    "index": "new_twitter"
  }
}'
reindex(body = body)

## End(Not run)
```

`scroll` *Scroll search function*

Description

Scroll search function

Usage

```
scroll(x, time_scroll = "1m", raw = FALSE, asdf = FALSE,
  stream_opts = list(), ...)

scroll_clear(x = NULL, all = FALSE, ...)
```

Arguments

`x`	(character) For `scroll`, a single scroll id; for `scroll_clear`, one or more scroll id's
`time_scroll`	(character) Specify how long a consistent view of the index should be maintained for scrolled search, e.g., "30s", "1m". See units-time.
`raw`	(logical) If `FALSE` (default), data is parsed to list. If `TRUE`, then raw JSON.
`asdf`	(logical) If `TRUE`, use `jsonlite::fromJSON()` to parse JSON directly to a data.frame. If `FALSE` (Default), list output is given.
`stream_opts`	(list) A list of options passed to `jsonlite::stream_out()` - Except that you can't pass `x` as that's the data that's streamed out, and pass a file path sinstead of a connection to `con`. `pagesize` param doesn't do much as that's more or less controlled by paging with ES.
`...`	Curl args passed on to `httr::POST()`
`all`	(logical) If `TRUE` (default) then all search contexts cleared. If `FALSE`, scroll id's must be passed to `x`

Value

`scroll()` returns a list, identical to what `Search()` returns. With attribute `scroll` that is the scroll value set via the `time_scroll` parameter

`scroll_clear()` returns a boolean (`TRUE` on success)

Scores

Scores will be the same for all documents that are returned from a scroll request. Dems da rules.

Inputs

Inputs to `scroll()` can be one of:

- list - This usually will be the output of `Search()`, but you could in theory make a list yourself with the appropriate elements
- character - A scroll ID - this is typically the scroll id output from a call to `Search()`, accessed like `res$\`_scroll_id\``

All other classes passed to `scroll()` will fail with message

Lists passed to `scroll()` without a `_scroll_id` element will trigger an error.

From lists output form `Search()` there should be an attribute ("scroll") that is the `scroll` value set in the `Search()` request - if that attribute is missing from the list, we'll attempt to use the `time_scroll` parameter value set in the `scroll()` function call

The output of `scroll()` has the scroll time value as an attribute so the output can be passed back into `scroll()` to continue.

Clear scroll

Search context are automatically removed when the scroll timeout has been exceeded. Keeping scrolls open has a cost, so scrolls should be explicitly cleared as soon as the scroll is not being used anymore using `scroll_clear`

Sliced scrolling

For scroll queries that return a lot of documents it is possible to split the scroll in multiple slices which can be consumed independently.

See the example in this man file.

Aggregations

If the request specifies aggregations, only the initial search response will contain the aggregations results.

References

https://www.elastic.co/guide/en/elasticsearch/reference/current/search-request-scroll.html

See Also

`Search()`

Examples

```
## Not run:
# Basic usage - can use across all indices
res <- Search(time_scroll="1m")
scroll(res)$`_scroll_id`
```

```
# use on a specific index - and specify a query
res <- Search(index = 'shakespeare', q="a*", time_scroll="1m")
res$`_scroll_id`

# Setting "sort=_doc" to turn off sorting of results - faster
res <- Search(index = 'shakespeare', q="a*", time_scroll="1m",
  body = '{"sort": ["_doc"]}')
res$`_scroll_id`

# Pass scroll_id to scroll function
scroll(res$`_scroll_id`)

# Get all results - one approach is to use a while loop
res <- Search(index = 'shakespeare', q="a*", time_scroll="5m",
  body = '{"sort": ["_doc"]}')
out <- list()
hits <- 1
while(hits != 0){
  res <- scroll(res$`_scroll_id`)
  hits <- length(res$hits$hits)
  if(hits > 0)
    out <- c(out, res$hits$hits)
}
length(out)
out[[1]]

# clear scroll
## individual scroll id
res <- Search(index = 'shakespeare', q="a*", time_scroll="5m",
  body = '{"sort": ["_doc"]}')
scroll_clear(res$`_scroll_id`)

## many scroll ids
res1 <- Search(index = 'shakespeare', q="c*", time_scroll="5m",
  body = '{"sort": ["_doc"]}')
res2 <- Search(index = 'shakespeare', q="d*", time_scroll="5m",
  body = '{"sort": ["_doc"]}')
nodes_stats(metric = "indices")$nodes[[1]]$indices$search$open_contexts
scroll_clear(c(res1$`_scroll_id`, res2$`_scroll_id`))
nodes_stats(metric = "indices")$nodes[[1]]$indices$search$open_contexts

## all scroll ids
res1 <- Search(index = 'shakespeare', q="f*", time_scroll="1m",
  body = '{"sort": ["_doc"]}')
res2 <- Search(index = 'shakespeare', q="g*", time_scroll="1m",
  body = '{"sort": ["_doc"]}')
res3 <- Search(index = 'shakespeare', q="k*", time_scroll="1m",
  body = '{"sort": ["_doc"]}')
scroll_clear(all = TRUE)

## sliced scrolling
body1 <- '{
  "slice": {
```

```
    "id": 0,
    "max": 2
  },
  "query": {
    "match" : {
      "text_entry" : "a*"
    }
  }
}'

body2 <- '{
  "slice": {
    "id": 1,
    "max": 2
  },
  "query": {
    "match" : {
      "text_entry" : "a*"
    }
  }
}'

res1 <- Search(index = 'shakespeare', time_scroll="1m", body = body1)
res2 <- Search(index = 'shakespeare', time_scroll="1m", body = body2)
scroll(res1$`_scroll_id`)
scroll(res2$`_scroll_id`)

out1 <- list()
hits <- 1
while(hits != 0){
  tmp1 <- scroll(res1$`_scroll_id`)
  hits <- length(tmp1$hits$hits)
  if(hits > 0)
    out1 <- c(out1, tmp1$hits$hits)
}

out2 <- list()
hits <- 1
while(hits != 0){
  tmp2 <- scroll(res2$`_scroll_id`)
  hits <- length(tmp2$hits$hits)
  if(hits > 0)
    out2 <- c(out2, tmp2$hits$hits)
}

c(
 lapply(out1, "[[", "_source"),
 lapply(out2, "[[", "_source")
)


# using jsonlite::stream_out
connect()
```

```
res <- Search(time_scroll = "1m")
file <- tempfile()
scroll(
  x = res$`_scroll_id`,
  stream_opts = list(file = file)
)
jsonlite::stream_in(file(file))
unlink(file)

## stream_out and while loop
connect()
(file <- tempfile())
res <- Search(index = "shakespeare", time_scroll = "5m",
  size = 1000, stream_opts = list(file = file))
while(!inherits(res, "warning")) {
  res <- tryCatch(scroll(
    x = res$`_scroll_id`,
    time_scroll = "5m",
    stream_opts = list(file = file)
  ), warning = function(w) w)
}
NROW(df <- jsonlite::stream_in(file(file)))
head(df)

## End(Not run)
```

Search *Full text search of Elasticsearch*

Description

Full text search of Elasticsearch

Usage

```
Search(index = NULL, type = NULL, q = NULL, df = NULL,
  analyzer = NULL, default_operator = NULL, explain = NULL,
  source = NULL, fields = NULL, sort = NULL, track_scores = NULL,
  timeout = NULL, terminate_after = NULL, from = NULL, size = NULL,
  search_type = NULL, lowercase_expanded_terms = NULL,
  analyze_wildcard = NULL, version = NULL, lenient = FALSE,
  body = list(), raw = FALSE, asdf = FALSE, time_scroll = NULL,
  search_path = "_search", stream_opts = list(), ...)
```

Arguments

index	Index name, one or more
type	Document type

`q` The query string (maps to the query_string query, see Query String Query for more details). See https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-query-string-query.html for documentation and examples.

`df` (character) The default field to use when no field prefix is defined within the query.

`analyzer` (character) The analyzer name to be used when analyzing the query string.

`default_operator` (character) The default operator to be used, can be `AND` or `OR`. Default: `OR`

`explain` (logical) For each hit, contain an explanation of how scoring of the hits was computed. Default: `FALSE`

`source` (logical) Set to `FALSE` to disable retrieval of the `_source` field. You can also retrieve part of the document by using `_source_include` & `_source_exclude` (see the `body` documentation for more details). You can also include a comma-delimited string of fields from the source document that you want back. See also the **fields** parameter

`fields` (character) The selective stored fields of the document to return for each hit. Not specifying any value will cause no fields to return. Note that in Elasticsearch v5 and greater, **fields** parameter has changed to **stored_fields**, which is not on by default. You can however, pass fields to **source** parameter

`sort` (character) Sorting to perform. Can either be in the form of fieldName, or `fieldName:asc`/`fieldName:desc`. The fieldName can either be an actual field within the document, or the special `_score` name to indicate sorting based on scores. There can be several sort parameters (order is important).

`track_scores` (logical) When sorting, set to `TRUE` in order to still track scores and return them as part of each hit.

`timeout` (numeric) A search timeout, bounding the search request to be executed within the specified time value and bail with the hits accumulated up to that point when expired. Default: no timeout.

`terminate_after` (numeric) The maximum number of documents to collect for each shard, upon reaching which the query execution will terminate early. If set, the response will have a boolean field terminated_early to indicate whether the query execution has actually terminated_early. Default: no terminate_after

`from` (character) The starting from index of the hits to return. Pass in as a character string to avoid problems with large number conversion to scientific notation. Default: 0

`size` (character) The number of hits to return. Pass in as a character string to avoid problems with large number conversion to scientific notation. Default: 10. The default maximum is 10,000 - however, you can change this default maximum by changing the `index.max_result_window` index level parameter.

`search_type` (character) The type of the search operation to perform. Can be `query_then_fetch` (default) or `dfs_query_then_fetch`. Types `scan` and `count` are deprecated. See http://bit.ly/19Am9xP for more details on the different types of search that can be performed.

lowercase_expanded_terms
: (logical) Should terms be automatically lowercased or not. Default: TRUE.

analyze_wildcard
: (logical) Should wildcard and prefix queries be analyzed or not. Default: FALSE.

version
: (logical) Print the document version with each document.

lenient
: If TRUE will cause format based failures (like providing text to a numeric field) to be ignored. Default: FALSE

body
: Query, either a list or json.

raw
: (logical) If FALSE (default), data is parsed to list. If TRUE, then raw JSON returned

asdf
: (logical) If TRUE, use fromJSON to parse JSON directly to a data.frame. If FALSE (Default), list output is given.

time_scroll
: (character) Specify how long a consistent view of the index should be maintained for scrolled search, e.g., "30s", "1m". See units-time

search_path
: (character) The path to use for searching. Default to _search, but in some cases you may already have that in the base url set using connect(), in which case you can set this to NULL

stream_opts
: (list) A list of options passed to stream_out - Except that you can't pass x as that's the data that's streamed out, and pass a file path instead of a connection to con. pagesize param doesn't do much as that's more or less controlled by paging with ES.

...
: Curl args passed on to POST

Details

This function name has the "S" capitalized to avoid conflict with the function base::search. I hate mixing cases, as I think it confuses users, but in this case it seems neccessary.

References

https://www.elastic.co/guide/en/elasticsearch/reference/current/search-search.html
https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html

See Also

Search_uri() Search_template() scroll() count() validate() fielddata()

Examples

```
## Not run:
if (!index_exists("shakespeare")) {
  shakespeare <- system.file("examples", "shakespeare_data.json", package = "elastic")
  invisible(docs_bulk(shakespeare))
}

# URI string queries
Search(index="shakespeare")
```

```
Search(index="shakespeare", type="act")
Search(index="shakespeare", type="scene")
Search(index="shakespeare", type="line")

## Return certain fields
if (gsub("\\.", "", ping()$version$number) < 500) {
  ### ES < v5
  Search(index="shakespeare", fields=c('play_name','speaker'))
} else {
  ### ES > v5
  Search(index="shakespeare", body = '{
   "_source": ["play_name", "speaker"]
  }')
}

## Search multiple indices
Search(index = "gbif")$hits$total
Search(index = "shakespeare")$hits$total
Search(index = c("gbif", "shakespeare"))$hits$total

## search_type
Search(index="shakespeare", search_type = "query_then_fetch")
Search(index="shakespeare", search_type = "dfs_query_then_fetch")
### search type "scan" is gone - use time_scroll instead
Search(index="shakespeare", time_scroll = "2m")
### search type "count" is gone - use size=0 instead
Search(index="shakespeare", size = 0)$hits$total

## search exists check
### use size set to 0 and terminate_after set to 1
### if there are > 0 hits, then there are matching documents
Search(index="shakespeare", type="act", size = 0, terminate_after = 1)

## sorting
### if ES >5, we need to make sure fielddata is turned on for a field
### before using it for sort
if (gsub("\\.", "", ping()$version$number) >= 500) {
  mapping_create("shakespeare", "act", update_all_types = TRUE, body = '{
    "properties": {
      "speaker": {
      "type":     "text",
      "fielddata": true
    }
  }
 }')
 Search(index="shakespeare", type="act", sort="speaker")
}

if (gsub("\\.", "", ping()$version$number) < 500) {
  Search(index="shakespeare", type="act", sort="speaker:desc",
    fields='speaker')
  Search(index="shakespeare", type="act",
    sort=c("speaker:desc","play_name:asc"), fields=c('speaker','play_name'))
```

```
}


## paging
if (gsub("\\.", "", ping()$version$number) < 500) {
  Search(index="shakespeare", size=1)$hits$hits
  Search(index="shakespeare", size=1, from=1)$hits$hits
}

## queries
### Search in all fields
Search(index="shakespeare", type="act", q="york")

### Searchin specific fields
Search(index="shakespeare", type="act", q="speaker:KING HENRY IV")$hits$total

### Exact phrase search by wrapping in quotes
Search(index="shakespeare", type="act", q='speaker:"KING HENRY IV"')$hits$total

### can specify operators between multiple words parenthetically
Search(index="shakespeare", type="act", q="speaker:(HENRY OR ARCHBISHOP)")$hits$total

### where the field line_number has no value (or is missing)
Search(index="shakespeare", q="_missing_:line_number")$hits$total

### where the field line_number has any non-null value
Search(index="shakespeare", q="_exists_:line_number")$hits$total

### wildcards, either * or ?
Search(index="shakespeare", q="*ay")$hits$total
Search(index="shakespeare", q="m?y")$hits$total

### regular expressions, wrapped in forward slashes
Search(index="shakespeare", q="text_entry:/[a-z]/")$hits$total

### fuzziness
Search(index="shakespeare", q="text_entry:ma~")$hits$total
Search(index="shakespeare", q="text_entry:the~2")$hits$total
Search(index="shakespeare", q="text_entry:the~1")$hits$total

### Proximity searches
Search(index="shakespeare", q='text_entry:"as hath"~5')$hits$total
Search(index="shakespeare", q='text_entry:"as hath"~10')$hits$total

### Ranges, here where line_id value is between 10 and 20
Search(index="shakespeare", q="line_id:[10 TO 20]")$hits$total

### Grouping
Search(index="shakespeare", q="(hath OR as) AND the")$hits$total

# Limit number of hits returned with the size parameter
Search(index="shakespeare", size=1)
```

```
# Give explanation of search in result
Search(index="shakespeare", size=1, explain=TRUE)

## terminate query after x documents found
## setting to 1 gives back one document for each shard
Search(index="shakespeare", terminate_after=1)
## or set to other number
Search(index="shakespeare", terminate_after=2)

## Get version number for each document
Search(index="shakespeare", version=TRUE, size=2)

## Get raw data
Search(index="shakespeare", type="scene", raw=TRUE)

## Curl options
library('httr')

### verbose
out <- Search(index="shakespeare", type="line", config = verbose())

### print progress
res <- Search(config = progress(), size = 5000)



# Query DSL searches - queries sent in the body of the request
## Pass in as an R list

### if ES >5, we need to make sure fielddata is turned on for a field
### before using it for aggregations
if (gsub("\\.", "", ping()$version$number) >= 500) {
  mapping_create("shakespeare", "act", update_all_types = TRUE, body = '{
    "properties": {
      "text_entry": {
        "type":     "text",
        "fielddata": true
     }
   }
 }')
 aggs <- list(aggs = list(stats = list(terms = list(field = "text_entry"))))
 Search(index="shakespeare", body=aggs)
}

### if ES >5, you don't need to worry about fielddata
if (gsub("\\.", "", ping()$version$number) < 500) {
   aggs <- list(aggs = list(stats = list(terms = list(field = "text_entry"))))
   Search(index="shakespeare", body=aggs)
}

## or pass in as json query with newlines, easy to read
aggs <- '{
    "aggs": {
```

```
        "stats" : {
            "terms" : {
                "field" : "text_entry"
            }
        }
    }
}'
Search(index="shakespeare", body=aggs)

## or pass in collapsed json string
aggs <- '{"aggs":{"stats":{"terms":{"field":"text_entry"}}}}'
Search(index="shakespeare", body=aggs)


## Aggregations
### Histograms
aggs <- '{
    "aggs": {
        "latbuckets" : {
           "histogram" : {
               "field" : "decimalLatitude",
               "interval" : 5
           }
        }
    }
}'
Search(index="gbif", body=aggs, size=0)

### Histograms w/ more options
aggs <- '{
    "aggs": {
        "latbuckets" : {
           "histogram" : {
               "field" : "decimalLatitude",
               "interval" : 5,
               "min_doc_count" : 0,
               "extended_bounds" : {
                   "min" : -90,
                   "max" : 90
               }
           }
        }
    }
}'
Search(index="gbif", body=aggs, size=0)

### Ordering the buckets by their doc_count - ascending:
aggs <- '{
    "aggs": {
        "latbuckets" : {
           "histogram" : {
               "field" : "decimalLatitude",
               "interval" : 5,
```

```
                "min_doc_count" : 0,
                "extended_bounds" : {
                    "min" : -90,
                    "max" : 90
                },
                "order" : {
                    "_count" : "desc"
                }
            }
        }
    }
}'
out <- Search(index="gbif", body=aggs, size=0)
lapply(out$aggregations$latbuckets$buckets, data.frame)

### By default, the buckets are returned as an ordered array. It is also possible to
### request the response as a hash instead keyed by the buckets keys:
aggs <- '{
    "aggs": {
        "latbuckets" : {
           "histogram" : {
               "field" : "decimalLatitude",
               "interval" : 10,
               "keyed" : true
           }
        }
    }
}'
Search(index="gbif", body=aggs, size=0)

# match query
match <- '{"query": {"match" : {"text_entry" : "Two Gentlemen"}}}'
Search(index="shakespeare", body=match)

# multi-match (multiple fields that is) query
mmatch <- '{"query": {"multi_match" : {"query" : "henry", "fields": ["text_entry","play_name"]}}}'
Search(index="shakespeare", body=mmatch)

# bool query
mmatch <- '{
 "query": {
   "bool" : {
     "must_not" : {
       "range" : {
         "speech_number" : {
           "from" : 1, "to": 5
}}}}}'
Search(index="shakespeare", body=mmatch)

# Boosting query
boost <- '{
 "query" : {
  "boosting" : {
```

```
      "positive" : {
          "term" : {
              "play_name" : "henry"
          }
      },
      "negative" : {
          "term" : {
              "text_entry" : "thou"
          }
      },
      "negative_boost" : 0.8
    }
 }
}'
Search(index="shakespeare", body=boost)

# Fuzzy query
## fuzzy query on numerics
fuzzy <- list(query = list(fuzzy = list(text_entry = "arms")))
Search(index="shakespeare", body=fuzzy)$hits$total
fuzzy <- list(query = list(fuzzy = list(text_entry = list(value = "arms", fuzziness = 4))))
Search(index="shakespeare", body=fuzzy)$hits$total

# geoshape query
## not working yets
geo <- list(query = list(geo_shape = list(location = list(shape = list(type = "envelope",
   coordinates = "[[2,10],[10,20]]")))))
geo <- '{
 "query": {
   "geo_shape": {
     "location": {
       "point": {
         "type": "envelope",
         "coordinates": [[2,0],[2.93,100]]
       }
     }
   }
 }
}'
# Search(index="gbifnewgeo", body=geo)

# range query
## with numeric
body <- list(query=list(range=list(decimalLongitude=list(gte=1, lte=3))))
Search('gbif', body=body)$hits$total

body <- list(query=list(range=list(decimalLongitude=list(gte=2.9, lte=10))))
Search('gbif', body=body)$hits$total

## with dates
body <- list(query=list(range=list(eventDate=list(gte="2012-01-01", lte="now"))))
Search('gbif', body=body)$hits$total
```

```
body <- list(query=list(range=list(eventDate=list(gte="2014-01-01", lte="now"))))
Search('gbif', body=body)$hits$total

# more like this query (more_like_this can be shortened to mlt)
body <- '{
 "query": {
   "more_like_this": {
     "fields": ["title"],
     "like_text": "and then",
     "min_term_freq": 1,
     "max_query_terms": 12
   }
 }
}'
Search('plos', body=body)$hits$total

body <- '{
 "query": {
   "more_like_this": {
     "fields": ["abstract","title"],
     "like_text": "cell",
     "min_term_freq": 1,
     "max_query_terms": 12
   }
 }
}'
Search('plos', body=body)$hits$total

# Highlighting
body <- '{
 "query": {
   "query_string": {
     "query" : "cell"
   }
 },
 "highlight": {
   "fields": {
     "title": {"number_of_fragments": 2}
   }
 }
}'
out <- Search('plos', 'article', body=body)
out$hits$total
sapply(out$hits$hits, function(x) x$`_source`$title[[1]])

### Common terms query
body <- '{
 "query" : {
   "common": {
      "body": {
           "query": "this is",
           "cutoff_frequency": 0.01
       }
```

```
    }
  }
}'
Search('shakespeare', 'line', body=body)

## Scrolling search - instead of paging
res <- Search(index = 'shakespeare', q="a*", time_scroll="1m")
scroll(res$`_scroll_id`)

res <- Search(index = 'shakespeare', q="a*", time_scroll="5m")
out <- list()
hits <- 1
while(hits != 0){
  res <- scroll(res$`_scroll_id`)
  hits <- length(res$hits$hits)
  if(hits > 0)
    out <- c(out, res$hits$hits)
}

### Sliced scrolling
#### For scroll queries that return a lot of documents it is possible to
#### split the scroll in multiple slices which can be consumed independently
body1 <- '{
  "slice": {
    "id": 0,
    "max": 2
  },
  "query": {
    "match" : {
      "text_entry" : "a*"
    }
  }
}'

body2 <- '{
  "slice": {
    "id": 1,
    "max": 2
  },
  "query": {
    "match" : {
      "text_entry" : "a*"
    }
  }
}'

res1 <- Search(index = 'shakespeare', time_scroll="1m", body = body1)
res2 <- Search(index = 'shakespeare', time_scroll="1m", body = body2)
scroll(res1$`_scroll_id`)
scroll(res2$`_scroll_id`)

out1 <- list()
hits <- 1
```

```
while(hits != 0){
  tmp1 <- scroll(res1$`_scroll_id`)
  hits <- length(tmp1$hits$hits)
  if(hits > 0)
    out1 <- c(out1, tmp1$hits$hits)
}

out2 <- list()
hits <- 1
while(hits != 0){
  tmp2 <- scroll(res2$`_scroll_id`)
  hits <- length(tmp2$hits$hits)
  if(hits > 0)
    out2 <- c(out2, tmp2$hits$hits)
}

c(
 lapply(out1, "[[", "_source"),
 lapply(out2, "[[", "_source")
)



# Using filters
## A bool filter
body <- '{
 "query":{
   "bool": {
     "must_not" : {
       "range" : {
         "year" : { "from" : 2011, "to" : 2012 }
       }
     }
   }
 }
}'
Search('gbif', body = body)$hits$total

## Geo filters - fun!
### Note that filers have many geospatial filter options, but queries
### have fewer, andrequire a geo_shape mapping

body <- '{
 "mappings": {
   "record": {
     "properties": {
         "location" : {"type" : "geo_point"}
      }
   }
 }
}'
index_recreate(index='gbifgeopoint', body=body)
path <- system.file("examples", "gbif_geopoint.json", package = "elastic")
```

```
invisible(docs_bulk(path))

### Points within a bounding box
body <- '{
 "query":{
   "bool" : {
     "must" : {
       "match_all" : {}
     },
     "filter":{
        "geo_bounding_box" : {
          "location" : {
            "top_left" : {
              "lat" : 60,
              "lon" : 1
            },
            "bottom_right" : {
              "lat" : 40,
              "lon" : 14
            }
          }
       }
     }
   }
 }
}'
out <- Search('gbifgeopoint', body = body, size = 300)
out$hits$total
do.call(rbind, lapply(out$hits$hits, function(x) x$`_source`$location))

### Points within distance of a point
body <- '{
"query": {
  "bool" : {
    "must" : {
      "match_all" : {}
    },
   "filter" : {
     "geo_distance" : {
       "distance" : "200km",
       "location" : {
         "lon" : 4,
         "lat" : 50
       }
     }
  }
}}}'
out <- Search('gbifgeopoint', body = body)
out$hits$total
do.call(rbind, lapply(out$hits$hits, function(x) x$`_source`$location))

### Points within distance range of a point
body <- '{
```

```
 "aggs":{
   "points_within_dist" : {
     "geo_distance" : {
        "field": "location",
        "origin" : "4, 50",
        "ranges": [
          {"from" : 200},
          {"to" : 400}
         ]
     }
   }
 }
}'
out <- Search('gbifgeopoint', body = body)
out$hits$total
do.call(rbind, lapply(out$hits$hits, function(x) x$`_source`$location))

### Points within a polygon
body <- '{
 "query":{
   "bool" : {
     "must" : {
       "match_all" : {}
     },
     "filter":{
        "geo_polygon" : {
          "location" : {
             "points" : [
               [80.0, -20.0], [-80.0, -20.0], [-80.0, 60.0], [40.0, 60.0], [80.0, -20.0]
             ]
           }
         }
       }
     }
   }
}'
out <- Search('gbifgeopoint', body = body)
out$hits$total
do.call(rbind, lapply(out$hits$hits, function(x) x$`_source`$location))

### Geoshape filters using queries instead of filters
#### Get data with geojson type location data loaded first
body <- '{
 "mappings": {
   "record": {
     "properties": {
         "location" : {"type" : "geo_shape"}
      }
   }
 }
}'
index_recreate(index='geoshape', body=body)
path <- system.file("examples", "gbif_geoshape.json", package = "elastic")
```

```
invisible(docs_bulk(path))

#### Get data with a square envelope, w/ point defining upper left and the other
#### defining the lower right
body <- '{
 "query":{
   "geo_shape" : {
     "location" : {
         "shape" : {
           "type": "envelope",
            "coordinates": [[-30, 50],[30, 0]]
         }
       }
     }
   }
}'
out <- Search('geoshape', body = body)
out$hits$total

#### Get data with a circle, w/ point defining center, and radius
body <- '{
 "query":{
   "geo_shape" : {
     "location" : {
         "shape" : {
           "type": "circle",
           "coordinates": [-10, 45],
           "radius": "2000km"
         }
       }
     }
   }
}'
out <- Search('geoshape', body = body)
out$hits$total

#### Use a polygon, w/ point defining center, and radius
body <- '{
 "query":{
   "geo_shape" : {
     "location" : {
         "shape" : {
           "type": "polygon",
           "coordinates":  [
            [ [80.0, -20.0], [-80.0, -20.0], [-80.0, 60.0], [40.0, 60.0], [80.0, -20.0] ]
           ]
         }
       }
     }
   }
}'
out <- Search('geoshape', body = body)
out$hits$total
```

```
# Geofilter with WKT
# format follows "BBOX (minlon, maxlon, maxlat, minlat)"
x <- '{
    "query": {
        "bool" : {
            "must" : {
                "match_all" : {}
            },
            "filter" : {
                "geo_bounding_box" : {
                    "pin.location" : {
                        "wkt" : "BBOX (1, 14, 60, 40)"
                    }
                }
            }
        }
    }
}'
out <- Search('gbifgeopoint', body = body)
out$hits$total



# Missing filter
if (gsub("\\.", "", ping()$version$number) < 500) {
  ### ES < v5
  body <- '{
   "query":{
     "constant_score" : {
       "filter" : {
         "missing" : { "field" : "play_name" }
       }
     }
    }
  }'
  Search("shakespeare", body = body)
} else {
  ### ES => v5
  body <- '{
   "query":{
     "bool" : {
       "must_not" : {
         "exists" : {
           "field" : "play_name"
         }
       }
     }
   }
  }'
  Search("shakespeare", body = body)
}
```

```
# prefix filter
body <- '{
 "query": {
   "bool": {
     "must": {
       "prefix" : {
         "speaker" : "we"
       }
     }
   }
 }
}'
x <- Search("shakespeare", body = body)
x$hits$total
vapply(x$hits$hits, "[[", "", c("_source", "speaker"))


# ids filter
if (gsub("\\.", "", ping()$version$number) < 500) {
  ### ES < v5
  body <- '{
   "query":{
     "bool": {
       "must": {
         "ids" : {
           "values": ["1","2","10","2000"]
         }
       }
     }
   }
  }'
  x <- Search("shakespeare", body = body)
  x$hits$total
  identical(
   c("1","2","10","2000"),
   vapply(x$hits$hits, "[[", "", "_id")
  )
} else {
  body <- '{
   "query":{
     "ids" : {
       "values": ["1","2","10","2000"]
     }
   }
  }'
  x <- Search("shakespeare", body = body)
  x$hits$total
  identical(
   c("2000","10","2","1"),
   vapply(x$hits$hits, "[[", "", "_id")
  )
}
```

```
# combined prefix and ids filters
if (gsub("\\.", "", ping()$version$number) < 500) {
  ### ES < v5
  body <- '{
   "query":{
     "bool" : {
       "should" : {
         "or": [{
           "ids" : {
             "values": ["1","2","3","10","2000"]
           }
         }, {
         "prefix" : {
           "speaker" : "we"
         }
        }
      ]
     }
    }
   }
  }'
  x <- Search("shakespeare", body = body)
  x$hits$total
} else {
  ### ES => v5
  body <- '{
   "query":{
     "bool" : {
       "should" : [
         {
           "ids" : {
             "values": ["1","2","3","10","2000"]
           }
         },
         {
           "prefix" : {
             "speaker" : "we"
           }
         }
      ]
     }
    }
  }'
  x <- Search("shakespeare", body = body)
  x$hits$total
}

# Suggestions
sugg <- '{
 "query" : {
    "match" : {
      "text_entry" : "late"
```

```
    }
 },
 "suggest" : {
   "sugg" : {
     "text" : "late",
     "term" : {
         "field" : "text_entry"
      }
    }
  }
}'
Search(index = "shakespeare", "line", body = sugg,
  asdf = TRUE, size = 0)$suggest$sugg$options



# stream data out using jsonlite::stream_out
file <- tempfile()
res <- Search("shakespeare", size = 1000, stream_opts = list(file = file))
head(df <- jsonlite::stream_in(file(file)))
NROW(df)
unlink(file)


## End(Not run)
```

`searchapis` *Search functions.*

Description

Search functions.

Details

Elasticsearch search APIs include the following functions:

- `Search()` - Search using the Query DSL via the body of the request.
- `Search_uri()` - Search using the URI search API only. This may be needed for servers that block POST requests for security, or maybe you don't need complicated requests, in which case URI only requests are suffice.
- `msearch()` - Multi Search - execute several search requests defined in a file passed to `msearch`
- `search_shards()` - Search shards.
- `count()` - Get counts for various searches.
- `explain()` - Computes a score explanation for a query and a specific document. This can give useful feedback whether a document matches or didn't match a specific query.
- `validate()` - Validate a search

- field_stats() - Search field statistics
- percolate() - Store queries into an index then, via the percolate API, define documents to retrieve these queries.

More will be added soon.

References

https://www.elastic.co/guide/en/elasticsearch/reference/current/search.html

---

search_shards *Search shards.*

Description

Search shards.

Usage

```
search_shards(index = NULL, raw = FALSE, routing = NULL,
  preference = NULL, local = NULL, ...)
```

Arguments

index	One or more indeces
raw	If TRUE (default), data is parsed to list. If FALSE, then raw JSON
routing	A character vector of routing values to take into account when determining which shards a request would be executed against.
preference	Controls a preference of which shard replicas to execute the search request on. By default, the operation is randomized between the shard replicas. See preference for a list of all acceptable values.
local	(logical) Whether to read the cluster state locally in order to determine where shards are allocated instead of using the Master node's cluster state.
...	Curl args passed on to httr::GET()

References

https://www.elastic.co/guide/en/elasticsearch/reference/current/search-shards.html

Examples

```
## Not run:
search_shards(index = "plos")
search_shards(index = c("plos","gbif"))
search_shards(index = "plos", preference='_primary')
search_shards(index = "plos", preference='_shards:2')

library('httr')
search_shards(index = "plos", config=verbose())

## End(Not run)
```

Search_template *Search or validate templates*

Description

Search or validate templates

Usage

```
Search_template(body = list(), raw = FALSE, ...)

Search_template_register(template, body = list(), raw = FALSE, ...)

Search_template_get(template, ...)

Search_template_delete(template, ...)

Search_template_render(body = list(), raw = FALSE, ...)
```

Arguments

body	Query, either a list or json.
raw	(logical) If FALSE (default), data is parsed to list. If TRUE, then raw JSON returned
...	Curl args passed on to httr::POST()
template	(character) a template name

Template search

With Search_template you can search with a template, using mustache templating. Added in Elasticsearch v1.1

Template render

With Search_template_render you validate a template without conducting the search. Added in Elasticsearch v2.0

Pre-registered templates

Register a template with `Search_template_register`. You can get the template with `Search_template_get` and delete the template with `Search_template_delete`

You can also pre-register search templates by storing them in the `config/scripts` directory, in a file using the .mustache extension. In order to execute the stored template, reference it by it's name under the template key, like `"file": "templateName", ...`

References

https://www.elastic.co/guide/en/elasticsearch/reference/current/search-template.html

See Also

`Search()`, `Search_uri()`

Examples

```
## Not run:
if (!index_exists("iris")) {
  invisible(docs_bulk(iris, "iris"))
}

body1 <- '{
  "inline" : {
    "query": { "match" : { "{{my_field}}" : "{{my_value}}" } },
    "size" : "{{my_size}}"
  },
  "params" : {
    "my_field" : "Species",
    "my_value" : "setosa",
    "my_size" : 3
  }
}'
Search_template(body = body1)

body2 <- '{
 "inline": {
   "query": {
      "match": {
          "Species": "{{query_string}}"
      }
   }
 },
 "params": {
   "query_string": "versicolor"
 }
}'
Search_template(body = body2)

# pass in a list
```

```
mylist <- list(
  inline = list(query = list(match = list(`{{my_field}}` = "{{my_value}}"))),
  params = list(my_field = "Species", my_value = "setosa", my_size = 3L)
)
Search_template(body = mylist)

## Validating templates w/ Search_template_render()
Search_template_render(body = body1)
Search_template_render(body = body2)

## pre-registered templates
### register a template
body3 <- '{
  "template": {
     "query": {
         "match": {
             "Species": "{{query_string}}"
         }
     }
   }
}'
Search_template_register('foobar', body = body3)

### get template
Search_template_get('foobar')

### use the template
body4 <- '{
 "id": "foobar",
   "params": {
      "query_string": "setosa"
  }
}'
Search_template(body = body4)

### delete the template
Search_template_delete('foobar')

## End(Not run)
```

Search_uri *Full text search of Elasticsearch with URI search*

Description

Full text search of Elasticsearch with URI search

Usage

```
Search_uri(index = NULL, type = NULL, q = NULL, df = NULL,
```

```
  analyzer = NULL, default_operator = NULL, explain = NULL,
  source = NULL, fields = NULL, sort = NULL, track_scores = NULL,
  timeout = NULL, terminate_after = NULL, from = NULL, size = NULL,
  search_type = NULL, lowercase_expanded_terms = NULL,
  analyze_wildcard = NULL, version = NULL, lenient = FALSE, raw = FALSE,
  asdf = FALSE, search_path = "_search", stream_opts = list(), ...)
```

Arguments

index	Index name, one or more
type	Document type
q	The query string (maps to the query_string query, see Query String Query for more details). See https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-query-string-query.html for documentation and examples.
df	(character) The default field to use when no field prefix is defined within the query.
analyzer	(character) The analyzer name to be used when analyzing the query string.
default_operator	(character) The default operator to be used, can be AND or OR. Default: OR
explain	(logical) For each hit, contain an explanation of how scoring of the hits was computed. Default: FALSE
source	(logical) Set to FALSE to disable retrieval of the _source field. You can also retrieve part of the document by using _source_include & _source_exclude (see the body documentation for more details). You can also include a comma-delimited string of fields from the source document that you want back. See also the **fields** parameter
fields	(character) The selective stored fields of the document to return for each hit. Not specifying any value will cause no fields to return. Note that in Elasticsearch v5 and greater, **fields** parameter has changed to **stored_fields**, which is not on by default. You can however, pass fields to **source** parameter
sort	(character) Sorting to perform. Can either be in the form of fieldName, or fieldName:asc/fieldName:desc. The fieldName can either be an actual field within the document, or the special _score name to indicate sorting based on scores. There can be several sort parameters (order is important).
track_scores	(logical) When sorting, set to TRUE in order to still track scores and return them as part of each hit.
timeout	(numeric) A search timeout, bounding the search request to be executed within the specified time value and bail with the hits accumulated up to that point when expired. Default: no timeout.
terminate_after	(numeric) The maximum number of documents to collect for each shard, upon reaching which the query execution will terminate early. If set, the response will have a boolean field terminated_early to indicate whether the query execution has actually terminated_early. Default: no terminate_after

from (character) The starting from index of the hits to return. Pass in as a character string to avoid problems with large number conversion to scientific notation. Default: 0

size (character) The number of hits to return. Pass in as a character string to avoid problems with large number conversion to scientific notation. Default: 10. The default maximum is 10,000 - however, you can change this default maximum by changing the index.max_result_window index level parameter.

search_type (character) The type of the search operation to perform. Can be query_then_fetch (default) or dfs_query_then_fetch. Types scan and count are deprecated. See http://bit.ly/19Am9xP for more details on the different types of search that can be performed.

lowercase_expanded_terms
(logical) Should terms be automatically lowercased or not. Default: TRUE.

analyze_wildcard
(logical) Should wildcard and prefix queries be analyzed or not. Default: FALSE.

version (logical) Print the document version with each document.

lenient If TRUE will cause format based failures (like providing text to a numeric field) to be ignored. Default: FALSE

raw (logical) If FALSE (default), data is parsed to list. If TRUE, then raw JSON returned

asdf (logical) If TRUE, use fromJSON to parse JSON directly to a data.frame. If FALSE (Default), list output is given.

search_path (character) The path to use for searching. Default to _search, but in some cases you may already have that in the base url set using connect(), in which case you can set this to NULL

stream_opts (list) A list of options passed to stream_out - Except that you can't pass x as that's the data that's streamed out, and pass a file path instead of a connection to con. pagesize param doesn't do much as that's more or less controlled by paging with ES.

... Curl args passed on to POST

See Also

fielddata()

Search() Search_template() count() fielddata()

Examples

```
## Not run:
# URI string queries
Search_uri(index="shakespeare")
Search_uri(index="shakespeare", type="act")
Search_uri(index="shakespeare", type="scene")
Search_uri(index="shakespeare", type="line")

## Return certain fields
```

```
if (gsub("\\.", "", ping()$version$number) < 500) {
  ### ES < v5
  Search_uri(index="shakespeare", fields=c('play_name','speaker'))
} else {
  ### ES > v5
  Search_uri(index="shakespeare", source=c('play_name','speaker'))
}

## Search many indices
Search_uri(index = "gbif")$hits$total
Search_uri(index = "shakespeare")$hits$total
Search_uri(index = c("gbif", "shakespeare"))$hits$total

## search_type
## NOTE: If you're in ES V5 or greater, see \code{?fielddata}
Search_uri(index="shakespeare", search_type = "query_then_fetch")
Search_uri(index="shakespeare", search_type = "dfs_query_then_fetch")
# Search_uri(index="shakespeare", search_type = "scan") # only when scrolling

## sorting
Search_uri(index="shakespeare", type="act", sort="text_entry")
Search_uri(index="shakespeare", type="act", sort="speaker:desc", fields='speaker')
Search_uri(index="shakespeare", type="act",
 sort=c("speaker:desc","play_name:asc"), fields=c('speaker','play_name'))

## paging
Search_uri(index="shakespeare", size=1, fields='text_entry')$hits$hits
Search_uri(index="shakespeare", size=1, from=1, fields='text_entry')$hits$hits

## queries
### Search in all fields
Search_uri(index="shakespeare", type="act", q="york")

### Searchin specific fields
Search_uri(index="shakespeare", type="act", q="speaker:KING HENRY IV")$hits$total

### Exact phrase search by wrapping in quotes
Search_uri(index="shakespeare", type="act", q='speaker:"KING HENRY IV"')$hits$total

### can specify operators between multiple words parenthetically
Search_uri(index="shakespeare", type="act", q="speaker:(HENRY OR ARCHBISHOP)")$hits$total

### where the field line_number has no value (or is missing)
Search_uri(index="shakespeare", q="_missing_:line_number")$hits$total

### where the field line_number has any non-null value
Search_uri(index="shakespeare", q="_exists_:line_number")$hits$total

### wildcards, either * or ?
Search_uri(index="shakespeare", q="*ay")$hits$total
Search_uri(index="shakespeare", q="m?y")$hits$total

### regular expressions, wrapped in forward slashes
```

```
Search_uri(index="shakespeare", q="text_entry:/[a-z]/")$hits$total

### fuzziness
Search_uri(index="shakespeare", q="text_entry:ma~")$hits$total
Search_uri(index="shakespeare", q="text_entry:the~2")$hits$total
Search_uri(index="shakespeare", q="text_entry:the~1")$hits$total

### Proximity searches
Search_uri(index="shakespeare", q='text_entry:"as hath"~5')$hits$total
Search_uri(index="shakespeare", q='text_entry:"as hath"~10')$hits$total

### Ranges, here where line_id value is between 10 and 20
Search_uri(index="shakespeare", q="line_id:[10 TO 20]")$hits$total

### Grouping
Search_uri(index="shakespeare", q="(hath OR as) AND the")$hits$total

# Limit number of hits returned with the size parameter
Search_uri(index="shakespeare", size=1)

# Give explanation of search in result
Search_uri(index="shakespeare", size=1, explain=TRUE)

## terminate query after x documents found
## setting to 1 gives back one document for each shard
Search_uri(index="shakespeare", terminate_after=1)
## or set to other number
Search_uri(index="shakespeare", terminate_after=2)

## Get version number for each document
Search_uri(index="shakespeare", version=TRUE, size=2)

## Get raw data
Search_uri(index="shakespeare", type="scene", raw=TRUE)

## Curl options
library('httr')

### verbose
out <- Search_uri(index="shakespeare", type="line", config=verbose())

### print progress
res <- Search_uri(config = progress(), size = 5000)

## End(Not run)
```

---

tasks *Elasticsearch tasks endpoints*

Description

Elasticsearch tasks endpoints

Usage

```
tasks(task_id = NULL, nodes = NULL, actions = NULL,
  parent_task_id = NULL, detailed = FALSE, group_by = NULL,
  wait_for_completion = FALSE, timeout = NULL, raw = FALSE, ...)

tasks_cancel(node_id = NULL, task_id = NULL, nodes = NULL,
  actions = NULL, parent_task_id = NULL, detailed = FALSE,
  group_by = NULL, wait_for_completion = FALSE, timeout = NULL,
  raw = FALSE, ...)
```

Arguments

task_id	a task id
nodes	(character) The nodes
actions	(character) Actions
parent_task_id	(character) A parent task ID
detailed	(character) get detailed results. Default: FALSE
group_by	(character) "nodes" (default, i.e., NULL) or "parents"
wait_for_completion	(logical) wait for completion. Default: FALSE
timeout	(integer) timeout time
raw	If TRUE (default), data is parsed to list. If FALSE, then raw JSON.
...	Curl args passed on to httr::GET() or httr::POST()
node_id	a node id

References

https://www.elastic.co/guide/en/elasticsearch/reference/current/tasks.html

Examples

```
## Not run:
connect()

tasks()
# tasks(parent_task_id = "1234")

# delete a task
# tasks_cancel()

## End(Not run)
```

termvectors *Termvectors*

Description

Termvectors

Usage

```
termvectors(index, type, id = NULL, body = list(), pretty = TRUE,
  field_statistics = TRUE, fields = NULL, offsets = TRUE, parent = NULL,
  payloads = TRUE, positions = TRUE, realtime = TRUE,
  preference = "random", routing = NULL, term_statistics = FALSE,
  version = NULL, version_type = NULL, ...)
```

Arguments

index	(character) The index in which the document resides.
type	(character) The type of the document.
id	(character) The id of the document, when not specified a doc param should be supplied.
body	(character) Define parameters and or supply a document to get termvectors for
pretty	(logical) pretty print. Default: TRUE
field_statistics	(character) Specifies if document count, sum of document frequencies and sum of total term frequencies should be returned. Default: TRUE
fields	(character) A comma-separated list of fields to return.
offsets	(character) Specifies if term offsets should be returned. Default: TRUE
parent	(character) Parent id of documents.
payloads	(character) Specifies if term payloads should be returned. Default: TRUE
positions	(character) Specifies if term positions should be returned. Default: TRUE
realtime	(character) Specifies if request is real-time as opposed to near-real-time (Default: TRUE).
preference	(character) Specify the node or shard the operation should be performed on (Default: random).
routing	(character) Specific routing value.
term_statistics	(character) Specifies if total term frequency and document frequency should be returned. Default: FALSE
version	(character) Explicit version number for concurrency control
version_type	(character) Specific version type, valid choices are: 'internal', 'external', 'external_gte', 'force'
...	Curl args passed on to httr::POST()

Details

Returns information and statistics on terms in the fields of a particular document. The document could be stored in the index or artificially provided by the user (Added in 1.4). Note that for documents stored in the index, this is a near realtime API as the term vectors are not available until the next refresh.

References

http://www.elastic.co/guide/en/elasticsearch/reference/current/docs-termvectors.html

Examples

```
## Not run:
connect()
if (!index_exists('plos')) {
  plosdat <- system.file("examples", "plos_data.json", package = "elastic")
  invisible(docs_bulk(plosdat))
}
if (!index_exists('omdb')) {
  omdb <- system.file("examples", "omdb.json", package = "elastic")
  invisible(docs_bulk(omdb))
}

body <- '{
  "fields" : ["title"],
  "offsets" : true,
  "positions" : true,
  "term_statistics" : true,
  "field_statistics" : true
}'
termvectors('plos', 'article', 29, body = body)

body <- '{
  "fields" : ["Plot"],
  "offsets" : true,
  "positions" : true,
  "term_statistics" : true,
  "field_statistics" : true
}'
termvectors('omdb', 'omdb', 'AVXdx8Eqg_0Z_tpMDyP_', body = body)

## End(Not run)
```

---

tokenizer_set *Tokenizer operations*

Description

Tokenizer operations

Usage

```
tokenizer_set(index, body, ...)
```

Arguments

index	(character) A character vector of index names
body	Query, either a list or json.
...	Curl options passed on to `httr::PUT()`

Author(s)

Scott Chamberlain myrmecocystus@gmail.com

Examples

```
## Not run:
# set tokenizer

## NGram tokenizer
body <- '{
        "settings" : {
             "analysis" : {
                 "analyzer" : {
                     "my_ngram_analyzer" : {
                         "tokenizer" : "my_ngram_tokenizer"
                     }
                 },
                 "tokenizer" : {
                     "my_ngram_tokenizer" : {
                         "type" : "nGram",
                         "min_gram" : "2",
                         "max_gram" : "3",
                         "token_chars": [ "letter", "digit" ]
                     }
                 }
             }
      }
}'
if (index_exists('test1')) index_delete('test1')
tokenizer_set(index = "test1", body=body)
index_analyze(text = "hello world", index = "test1",
  analyzer='my_ngram_analyzer')

## End(Not run)
```

units-distance *Distance units*

Description

Wherever distances need to be specified, such as the distance parameter in the Geo Distance Filter), the default unit if none is specified is the meter. Distances can be specified in other units, such as "1km" or "2mi" (2 miles).

Details

mi or miles	Mile
yd or yards	Yard
ft or feet	Feet
in or inch	Inch
km or kilometers	Kilometer
m or meters	Meter
cm or centimeters	Centimeter
mm or millimeters	Millimeter
NM, nmi or nauticalmiles	Nautical mile

The precision parameter in the Geohash Cell Filter accepts distances with the above units, but if no unit is specified, then the precision is interpreted as the length of the geohash.

See Also

units-time

units-time *Time units*

Description

Whenever durations need to be specified, eg for a timeout parameter, the duration can be specified as a whole number representing time in milliseconds, or as a time value like 2d for 2 days. The supported units are:

Details

y	Year
M	Month
w	Week
d	Day

h	Hour
m	Minute
s	Second

See Also

units-distance

`validate` *Validate a search*

Description

Validate a search

Usage

```
validate(index, type = NULL, ...)
```

Arguments

`index`	Index name. Required.
`type`	Document type. Optional.
`...`	Additional args passed on to `Search()`

See Also

`Search()`

Examples

```
## Not run:
if (!index_exists("twitter")) index_create("twitter")
docs_create('twitter', type='tweet', id=1, body = list(
   "user" = "foobar",
   "post_date" = "2014-01-03",
   "message" = "trying out Elasticsearch"
 )
)
validate("twitter", q='user:foobar')
validate("twitter", "tweet", q='user:foobar')

body <- '{
"query" : {
  "bool" : {
```

```
    "must" : {
      "query_string" : {
        "query" : "*:*"
      }
    },
    "filter" : {
      "term" : { "user" : "kimchy" }
    }
  }
}
}'
validate("twitter", body = body)

## End(Not run)
```

Index