

# Package ‘elastic’

November 9, 2016

**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 'APIs' for 'indices', documents, nodes, clusters, an interface to the cat 'API', and more.

**Version** 0.7.8

**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 (>= 5.0.1), testthat, knitr, covr

**RoxygenNote** 5.0.1

**NeedsCompilation** no

**Author** Scott Chamberlain [aut, cre]

**Maintainer** Scott Chamberlain <[myrmecocystus@gmail.com](mailto:myrmecocystus@gmail.com)>

**Repository** CRAN

**Date/Publication** 2016-11-09 18:28:49

## R topics documented:

alias . . . . .	2
cat . . . . .	4
cluster . . . . .	6
connect . . . . .	9

count . . . . .	11
docs_bulk . . . . .	12
docs_bulk_prep . . . . .	15
docs_create . . . . .	18
docs_delete . . . . .	19
docs_get . . . . .	20
docs_mget . . . . .	21
docs_update . . . . .	23
documents . . . . .	25
elastic . . . . .	26
elastic-defunct . . . . .	27
explain . . . . .	28
field_stats . . . . .	29
index . . . . .	30
info . . . . .	36
mapping . . . . .	37
msearch . . . . .	39
mtermvectors . . . . .	40
nodes . . . . .	43
percolate . . . . .	45
ping . . . . .	48
preference . . . . .	48
scroll . . . . .	49
Search . . . . .	52
searchapis . . . . .	67
search_shards . . . . .	68
Search_template . . . . .	69
Search_uri . . . . .	71
termvectors . . . . .	75
tokenizer_set . . . . .	77
units-distance . . . . .	78
units-time . . . . .	79
validate . . . . .	79
<b>Index</b>	<b>81</b>

---

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 <a href="#">POST</a>
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 <a href="#">GET</a>
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 <code>fielddata</code>

### 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 `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)

```

**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 <b>elastic</b>
headers	Either an object of class request or a list that can be coerced to an object of class request via <a href="#">add_headers</a> . These headers are used in all requests. To use headers in individual requests and not others, pass in headers using <a href="#">add_headers</a> 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(list(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, ...)
```

**Arguments**

x	A data.frame or path to a file to load in the bulk API
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 than 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.
...	Pass on curl options to <a href="#">POST</a>

## 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 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.

## 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.

## See Also

[docs\\_bulk\\_prep](#)

## Examples

```
## Not run:
plosdat <- system.file("examples", "plos_data.json", package = "elastic")
docs_bulk(plosdat)
aliases_get()
index_delete(index='plos')
aliases_get()

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

```

# 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)

## 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)
```

## 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

**Value**

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

**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)

## End(Not run)

```

---

docs\_create                      *Create a document*

---

### Description

Create a document

### Usage

```
docs_create(index, type, id, 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. Required
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 <a href="#">PUT</a>
...	Further args to query DSL

### References

[https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-index\\_.html](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))
}

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)

## 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 <a href="#">DELETE</a>
...	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
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 <a href="http://bit.ly/2efmYiE">http://bit.ly/2efmYiE</a> 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 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 <a href="http://bit.ly/2ezW9Zr">http://bit.ly/2ezW9Zr</a> 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 <a href="http://bit.ly/2fbqkZ1">http://bit.ly/2fbqkZ1</a> 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 <a href="#">POST</a>
...	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](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](mailto: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	<i>Explain a search query.</i>
---------	--------------------------------

---

**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 <code>_source</code> of the document explained. You can also retrieve part of the document by using <code>source_include</code> & <code>source_exclude</code> (see Get API for more details). This matches the <code>_source</code> 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 <code>query_string</code> query).
df	The default field to use when no field prefix is defined within the query. Defaults to <code>_all</code> field.
analyzer	The analyzer name to be used when analyzing the query string. Defaults to the analyzer of the <code>_all</code> 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 <a href="#">GET</a>

## References

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

## Examples

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

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

## End(Not run)
```

---

field_stats	<i>Search field statistics</i>
-------------	--------------------------------

---

## 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. Required.
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 [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.

### See Also

[Search\\_uri](#) [Search](#) [msearch](#)

### Examples

```
## Not run:
connect()
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_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 character vector of features. One or more of settings, mappings, warmers 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 <a href="#">POST</a> , <a href="#">GET</a> , <a href="#">PUT</a> , <a href="#">HEAD</a> , or <a href="#">DELETE</a>
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. [1.4.0.Beta1]
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/elasticsearch/reference/current/indices-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')
index_get(index='shakespeare', features=c('settings', 'mappings'))
index_get(index='shakespeare', features='aliases')
index_get(index='shakespeare', features='warmers')

# 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 = "indices")
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
index_optimize('plos')
index_optimize(c('plos','gbif'))

# Upgrade one or more indices to the latest format. The upgrade process converts any
# segments written with previous formats.
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 thou", 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)

```

**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, ...)

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.

... Curl options passed on to [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="string"),
  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": "string" },
      "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("string")
)))
# 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")
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
file <- system.file("examples", "gbif_geopoint.json", package = "elastic")
docs_bulk(file)
body <- '{
  "properties" : {
    "location" : { "type" : "geo_point" }
  }
}'
mapping_create("gbifgeopoint", "record", body = body)

## 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 <a href="#">fromJSON</a> to parse JSON directly to a data.frame. If FALSE (Default), list output is given.
...	Curl args passed on to <a href="#">POST</a>

## 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 <a href="#">POST</a>

## 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
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 <a href="#">GET</a>
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 `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')
nodes_hot_threads()

## 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,
  ...)

```

```

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 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 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.

... 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.

## References

<https://www.elastic.co/guide/en/elasticsearch/reference/current/search-percolate.html>

## Examples

```
## Not run:
# typical usage
## create an index first
```

```
index_delete("myindex")
mapping <- '{
  "mappings": {
    "mytype": {
      "properties": {
        "message": {
          "type": "string"
        },
        "name": {
          "type": "string"
        }
      }
    }
  }
}'
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

## End(Not run)
```

---

ping	<i>Ping an Elasticsearch server.</i>
------	--------------------------------------

---

### Description

Ping an Elasticsearch server.

### Usage

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

### Arguments

```
...          Curl args passed on to GET
```

### See Also

[connect](#)

### Examples

```
## Not run:
ping()

## End(Not run)
```

---

preference	<i>Preferences.</i>
------------	---------------------

---

### 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.

---

 scroll

*Scroll search function*


---

**Description**

Scroll search function

**Usage**

```
scroll(scroll_id, scroll = "1m", raw = FALSE, ...)
```

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

**Arguments**

<code>scroll_id</code>	(character) For <code>scroll</code> , a single scroll id; for <code>scroll_clear</code> , one or more scroll id's
<code>scroll</code>	(character) Specify how long a consistent view of the index should be maintained for scrolled search, e.g., "30s", "1m". See <a href="#">units-time</a> .
<code>raw</code>	(logical) If TRUE (default), data is parsed to list. If FALSE, then raw JSON.
<code>...</code>	Curl args passed on to <a href="#">POST</a>
<code>all</code>	(logical) If TRUE (default) then all search contexts cleared. If FALSE, scroll id's must be passed to <code>scroll_id</code>

**Details**

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

## 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

See the example in this man file.

## References

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

## See Also

[Search](#)

## Examples

```
## Not run:
# Get a scroll_id
res <- Search(index = 'shakespeare', q="a*", scroll="1m")
res$`_scroll_id`

# Setting search_type=scan turns off sorting of results, is faster
res <- Search(index = 'shakespeare', q="a*", scroll="1m", search_type = "scan")
res$`_scroll_id`

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

# Get all results - one approach is to use a while loop
res <- Search(index = 'shakespeare', q="a*", scroll="5m", search_type = "scan")
out <- list()
hits <- 1
while(hits != 0){
  res <- scroll(scroll_id = 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*", scroll="5m", search_type = "scan")
scroll_clear(scroll_id = res$`_scroll_id`)

## many scroll ids
```

```

res1 <- Search(index = 'shakespeare', q="c*", scroll="5m", search_type = "scan")
res2 <- Search(index = 'shakespeare', q="d*", scroll="5m", search_type = "scan")
nodes_stats(metric = "indices")$nodes[[1]]$indices$search$open_contexts
scroll_clear(scroll_id = 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*", scroll="1m", search_type = "scan")
res2 <- Search(index = 'shakespeare', q="g*", scroll="1m", search_type = "scan")
res3 <- Search(index = 'shakespeare', q="k*", scroll="1m", search_type = "scan")
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', scroll="1m", body = body1)
res2 <- Search(index = 'shakespeare', scroll="1m", body = body2)
scroll(scroll_id = res1$`_scroll_id`)
scroll(scroll_id = res2$`_scroll_id`)

out1 <- list()
hits <- 1
while(hits != 0){
  tmp1 <- scroll(scroll_id = 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(scroll_id = res2$`_scroll_id`)
  hits <- length(tmp2$hits$hits)
  if(hits > 0)
    out2 <- c(out2, tmp2$hits$hits)
}

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

## 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 = FALSE, lenient = FALSE,
  body = list(), raw = FALSE, asdf = FALSE, scroll = NULL,
  search_path = "_search", ...)

```

## Arguments

index	Index name, one or more
type	Document type
q	The query string (maps to the <code>query_string</code> query, see <a href="https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-query-string-query.html">Query String Query</a> for more details). See <a href="https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-query-string-query.html">https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-query-string-query.html</a> 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 <code>_source</code> field. You can also retrieve part of the document by using <code>_source_include</code> & <code>_source_exclude</code> (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 <b>fields</b> 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, <b>fields</b> parameter has changed to <b>stored_fields</b> , which is not on by default. You can however, pass fields to <b>source</b> parameter
sort	(character) Sorting to perform. Can either be in the form of <code>fieldName</code> , or <code>fieldName:asc/fieldName:desc</code> . The <code>fieldName</code> can either be an actual field within the document, or the special <code>_score</code> 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 <code>terminated_early</code> to indicate whether the query execution has actually terminated_early. Defaults to no <code>terminate_after</code> .
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 <code>index.max_result_window</code> index level parameter.
search_type	(character) The type of the search operation to perform. Can be <code>query_then_fetch</code> (default) or <code>dfs_query_then_fetch</code> . Types <code>scan</code> and <code>count</code> are deprecated. See <a href="http://bit.ly/19Am9xP">http://bit.ly/19Am9xP</a> 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 <code>fromJSON</code> to parse JSON directly to a data.frame. If FALSE (Default), list output is given.
scroll	(character) Specify how long a consistent view of the index should be maintained for scrolled search, e.g., "30s", "1m". See <code>units-time</code> .
search_path	(character) The path to use for searching. Default to <code>_search</code> , but in some cases you may already have that in the base url set using <code>connect</code> , in which case you can set this to NULL
...	Curl args passed on to <code>POST</code>

### 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 necessary.

### 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](#)

### Examples

```
## Not run:
if (!index_exists("shakespeare")) {
  shakespeare <- system.file("examples", "shakespeare_data.json", package = "elastic")
  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 scroll instead
Search(index="shakespeare", 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
Search(index="shakespeare", type="act", sort="text_entry")
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, fields='text_entry')$hits$hits
  Search(index="shakespeare", size=1, from=1, fields='text_entry')$hits$hits
}

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

### Search in 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 debugging
library('httr')
out <- Search(index="shakespeare", type="line", config=verbose())

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

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(speech_number = 7)))
Search(index="shakespeare", body=fuzzy)$hits$total
fuzzy <- list(query = list(fuzzy = list(speech_number = list(value = 7, fuzziness = 4))))
Search(index="shakespeare", body=fuzzy)$hits$total

### fuzzy query on date/time
fuzzy <- list(query = list(fuzzy = list(eventDate = list(value = "2014-01-29T23:00:00.000",
  fuzziness = "1d"))))
out <- Search(index="gbif", body=fuzzy, fields="eventDate")
out$hits$total
sapply(out$hits$hits, function(x) x$fields$eventDate) ## sweet!!!

# 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
Search('shakespeare', q="a*")$hits$total
res <- Search(index = 'shakespeare', q="a*", scroll="1m")
res <- Search(index = 'shakespeare', q="a*", scroll="1m",
  search_type = "scan")
scroll(scroll_id = res$`_scroll_id`)

res <- Search(index = 'shakespeare', q="a*", scroll="5m",
  search_type = "scan")
out <- list()
hits <- 1
while(hits != 0){
  res <- scroll(scroll_id = 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', scroll="1m", body = body1)
res2 <- Search(index = 'shakespeare', scroll="1m", body = body2)
scroll(scroll_id = res1$`_scroll_id`)
scroll(scroll_id = res2$`_scroll_id`)

out1 <- list()
hits <- 1
while(hits != 0){
  tmp1 <- scroll(scroll_id = 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(scroll_id = 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 filters have many geospatial filter options, but queries
### have fewer, and require 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")
docs_bulk(path)

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

### Points within distance of a point
body <- '{
  "query":{
    "filtered":{
      "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 <- '{
  "query":{
    "bool" : {
      "must" : {
        "match_all" : {}
      },
      "filter":{
        "geo_distance_range" : {
          "from" : "200km",
          "to" : "400km",
          "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 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")
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

# Missing filter
body <- '{
  "query":{
    "constant_score" : {
      "filter" : {
        "missing" : { "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
body <- '{
  "query":{
    "bool": {
      "must": {
        "ids" : {
          "values": ["1","2","3","10","2000"]
        }
      }
    }
  }
}'

```

```

}'
x <- Search("shakespeare", body = body)
x$hits$total
identical(
  c("1","2","3","10","2000"),
  vapply(x$hits$hits, "[", "", "_id")
)

# combined prefix and ids filters
body <- '{
  "query":{
    "bool" : {
      "should" : {
        "or": [
          {
            "ids" : {
              "values": ["1","2","3","10","2000"]
            }
          },
          {
            "prefix" : {
              "speaker" : "we"
            }
          }
        ]
      }
    }
  }
}'
x <- Search("shakespeare", body = body)
x$hits$total

## 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	<i>Search shards.</i>
---------------	-----------------------

---

## Description

Search shards.

## Usage

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

## Arguments

index	One or more indexes
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 <a href="#">preference</a> 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 <a href="#">GET</a>

## 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	<i>Search or validate templates</i>
-----------------	-------------------------------------

---

**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 <a href="#">POST</a>
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 its name under the template key, like `"file": "templateName", ...`

### 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 = FALSE, lenient = FALSE,
  raw = FALSE, asdf = FALSE, search_path = "_search", ...)

```

**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 <a href="https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-query-string-query.html">https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-query-string-query.html</a> 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 <b>fields</b> 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, <b>fields</b> parameter has changed to <b>stored_fields</b> , which is not on by default. You can however, pass fields to <b>source</b> 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. Defaults to 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 <a href="http://bit.ly/19Am9xP">http://bit.ly/19Am9xP</a> 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 <a href="#">fromJSON</a> 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 <a href="#">connect</a> , in which case you can set this to NULL
...	Curl args passed on to <a href="#">POST</a>

**See Also**

[Search Search\\_template count](#)

**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
Search_uri(index="shakespeare", fields=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
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")

### Search in 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 debugging
library('httr')
out <- Search_uri(index="shakespeare", type="line", config=verbose())

## 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 <a href="#">POST</a>

### 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	<i>Tokenizer operations</i>
---------------	-----------------------------

---

## 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 <a href="#">PUT</a>

## 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**

<code>index</code>	Index name. Required.
<code>type</code>	Document type. Optional.
<code>...</code>	Additional args passed on to <a href="#">Search</a>

**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

add\_headers, [10](#)  
alias, [2](#)  
alias\_create (alias), [2](#)  
alias\_delete (alias), [2](#)  
alias\_exists (alias), [2](#)  
alias\_get (alias), [2](#)  
aliases\_get (alias), [2](#)

cat, [4](#), [5](#), [44](#)  
cat\_, [5](#)  
cat\_ (cat), [4](#)  
cat\_aliases (cat), [4](#)  
cat\_allocation (cat), [4](#)  
cat\_count (cat), [4](#)  
cat\_fielddata (cat), [4](#)  
cat\_health (cat), [4](#)  
cat\_indices (cat), [4](#)  
cat\_master (cat), [4](#)  
cat\_nodeattrs (cat), [4](#)  
cat\_nodes (cat), [4](#)  
cat\_pending\_tasks (cat), [4](#)  
cat\_plugins (cat), [4](#)  
cat\_recovery (cat), [4](#)  
cat\_segments (cat), [4](#)  
cat\_shards (cat), [4](#)  
cat\_thread\_pool (cat), [4](#)  
cluster, [6](#)  
cluster\_health (cluster), [6](#)  
cluster\_pending\_tasks (cluster), [6](#)  
cluster\_reroute (cluster), [6](#)  
cluster\_settings (cluster), [6](#)  
cluster\_state (cluster), [6](#)  
cluster\_stats (cluster), [6](#)  
connect, [9](#), [10](#), [26](#), [48](#), [54](#), [73](#)  
connection, [10](#)  
connection (connect), [9](#)  
count, [11](#), [54](#), [68](#), [73](#)

DELETE, [19](#), [32](#)  
docs\_bulk, [12](#), [16](#), [25](#), [39](#)  
docs\_bulk\_prep, [13](#), [15](#)  
docs\_create, [18](#), [25](#)  
docs\_delete, [19](#), [25](#)  
docs\_get, [20](#), [25](#)  
docs\_mget, [21](#), [25](#)  
docs\_update, [23](#)  
documents, [25](#)

elastic, [26](#)  
elastic-defunct, [27](#)  
elastic-package (elastic), [26](#)  
explain, [28](#), [68](#)

field\_mapping\_get (mapping), [37](#)  
field\_stats, [29](#), [68](#)  
fromJSON, [30](#), [39](#), [54](#), [73](#)

GET, [5](#), [29](#), [32](#), [44](#), [48](#), [68](#)

HEAD, [32](#), [37](#)

index, [30](#)  
index\_analyze (index), [30](#)  
index\_clear\_cache (index), [30](#)  
index\_close (index), [30](#)  
index\_create (index), [30](#)  
index\_delete (index), [30](#)  
index\_exists (index), [30](#)  
index\_flush (index), [30](#)  
index\_get (index), [30](#)  
index\_open (index), [30](#)  
index\_optimize (index), [30](#)  
index\_recovery, [33](#)  
index\_recovery (index), [30](#)  
index\_recreate (index), [30](#)  
index\_segments (index), [30](#)  
index\_settings (index), [30](#)  
index\_settings\_update (index), [30](#)  
index\_stats (index), [30](#)  
index\_status, [27](#)  
index\_upgrade (index), [30](#)

info, 36

mapping, 37

mapping\_create (mapping), 37

mapping\_delete, 27

mapping\_get (mapping), 37

mlt, 27

msearch, 30, 39, 68

mtermvectors, 40

nodes, 43

nodes\_hot\_threads, 44

nodes\_hot\_threads (nodes), 43

nodes\_info (nodes), 43

nodes\_shutdown, 27

nodes\_stats (nodes), 43

percolate, 45, 68

percolate\_count (percolate), 45

percolate\_delete (percolate), 45

percolate\_list (percolate), 45

percolate\_match (percolate), 45

percolate\_register (percolate), 45

ping, 10, 48

POST, 3, 12, 24, 30, 32, 39, 41, 49, 54, 69, 73, 76

preference, 48, 68

PUT, 18, 32, 77

scroll, 49, 54

scroll\_clear (scroll), 49

Search, 11, 26, 27, 30, 40, 46, 50, 52, 67, 70, 73, 79, 80

search\_shards, 68, 68

Search\_template, 54, 69, 73

Search\_template\_delete (Search\_template), 69

Search\_template\_get (Search\_template), 69

Search\_template\_register (Search\_template), 69

Search\_template\_render (Search\_template), 69

Search\_uri, 11, 27, 30, 40, 54, 67, 70, 71

searchapis, 67

termvectors, 75

tokenizer\_set, 77

type\_exists (mapping), 37

units-distance, 78

units-time, 79

validate, 54, 68, 79