pyelasticsearch Documentation

Release 1.4

pyelasticsearch authors

Mar 07, 2017

Contents

pyelasticsearch is a clean, future-proof, high-scale API to elasticsearch. It provides...

- Transparent conversion of Python data types to and from JSON, including datetimes and the arbitrary-precision Decimal type
- Translation of HTTP failure status codes into exceptions
- Connection pooling
- HTTP basic auth and HTTPS support
- Load balancing across nodes in a cluster
- Failed-node marking to avoid downed nodes for a period
- Optional automatic retrying of failed requests
- Thread safety
- Loosely coupled design, letting you customize things like JSON encoding and bulk indexing

For more on our philosophy and history, see *Comparison with elasticsearch-py, the "Official Client"*.

CHAPTER 1

A Taste of the API

Make a pooling, balancing, all-singing, all-dancing connection object:

```
>>> from pyelasticsearch import ElasticSearch
>>> es = ElasticSearch('http://localhost:9200/')
```

Index a document:

```
>>> es.index('contacts',
...          'person',
...          {'name': 'Joe Tester', 'age': 25, 'title': 'QA Master'},
...           id=1)
{u'_type': u'person', u'_id': u'1', u'ok': True, u'_version': 1, u'_index': u'contacts
↪'}
```

Index a couple more documents, this time in a single request using the bulk-indexing API:

```
>>>  docs = [{'id': 2, 'name': 'Jessica Coder', 'age': 32, 'title': 'Programmer'},
...          {'id': 3, 'name': 'Freddy Tester', 'age': 29, 'title': 'Office Assistant
↪'}]
>>>  es.bulk((es.index_op(doc, id=doc.pop('id')) for doc in docs),
...          index='contacts',
...          doc_type='person')
```

If we had many documents and wanted to chunk them for performance, *bulk_chunks()* would easily rise to the task, dividing either at a certain number of documents per batch or, for curated platforms like Google App Engine, at a certain number of bytes. Thanks to the decoupled design, you can even substitute your own batching function if you have unusual needs. Bulk indexing is the most demanding ES task in most applications, so we provide very thorough tools for representing operations, optimizing wire traffic, and dealing with errors. See *bulk()* for more.

Refresh the index to pick up the latest:

```
>>> es.refresh('contacts')
{u'ok': True, u'_shards': {u'successful': 5, u'failed': 0, u'total': 10}}
```

Get just Jessica's document:

```
>>> es.get('contacts', 'person', 2)
{u'_id': u'2',
 u'_index': u'contacts',
 u'_source': {u'age': 32, u'name': u'Jessica Coder', u'title': u'Programmer'},
 u'_type': u'person',
 u'_version': 1,
 u'exists': True}
```

Perform a simple search:

```
>>> es.search('name:joe OR name:freddy', index='contacts')
{u'_shards': {u'failed': 0, u'successful': 42, u'total': 42},
 u'hits': {u'hits': [{u'_id': u'1',
                      u'_index': u'contacts',
                      u'_score': 0.028130024999999999,
                      u'_source': {u'age': 25,
                                   u'name': u'Joe Tester',
                                   u'title': u'QA Master'},
                      u'_type': u'person'},
                     {u'_id': u'3',
                      u'_index': u'contacts',
                      u'_score': 0.028130024999999999,
                      u'_source': {u'age': 29,
                                   u'name': u'Freddy Tester',
                                   u'title': u'Office Assistant'},
                      u'_type': u'person'}],
           u'max_score': 0.028130024999999999,
           u'total': 2},
 u'timed_out': False,
 u'took': 4}
```

Perform a search using the elasticsearch query DSL:

```
>>> query = {
...     'query': {
...         'filtered': {
...             'query': {
...                 'query_string': {'query': 'name:tester'}
...             },
...             'filter': {
...                 'range': {
...                     'age': {
...                         'from': 27,
...                         'to': 37,
...                     },
...                 },
...             },
...         },
...     },
... }
>>> es.search(query, index='contacts')
{u'_shards': {u'failed': 0, u'successful': 42, u'total': 42},
 u'hits': {u'hits': [{u'_id': u'3',
                      u'_index': u'contacts',
                      u'_score': 0.19178301,
                      u'_source': {u'age': 29,
                                   u'name': u'Freddy Tester',
                                   u'title': u'Office Assistant'},
```

```
                    u'_type': u'person'}],
          u'max_score': 0.19178301,
          u'total': 1},
 u'timed_out': False,
 u'took': 2}
```

Delete the index:

```
>>> es.delete_index('contacts')
{u'acknowledged': True, u'ok': True}
```

For more, see the full *API Documentation*.

CHAPTER 2

Contents

Features

JSON Conversion

pyelasticsearch converts transparently between Python datastructures and JSON.

- In request bodies, all the standard conversions are made: strings, numeric types, nulls, etc.
- We convert `datetime` and `date` instances to the format ES understands: `2012-02-23T14:26:01`. `date` objects are taken to represent midnight on their day.
- Python sets are converted to ES lists.

You can customize JSON conversion by setting the *`json_encoder`* attribute on an ElasticSearch object.

Connection Pooling

Connection pooling saves setting up a whole new TCP connection for each ES request, dropping latency by an order of magnitude. The ElasticSearch object is thread-safe; to take best advantage of connection pooling, create one instance, and share it among all threads. At most, the object will hold a number of connections to each node equal to the number of threads.

Load-balancing and Failover

An *`ElasticSearch`* object can take a list of node URLs on construction. This lets us balance load and maintain availability when nodes go down: pyelasticsearch will randomly choose a server URL for each request. If a node fails to respond before a timeout period elapses, it is assumed down and not tried again for awhile. Meanwhile, pyelasticsearch will retry the request on a different node if `max_retries` was set to something greater than zero at construction. If *all* nodes are marked as down, pyelasticsearch will loosen its standards and try sending requests to them, marking them alive if they respond.

Forward-Compatibility Kwargs

All methods that correspond to ES calls take an arbitrary set of kwargs that can be used to pass query string parameters directly to ES. Certain kwargs (called out by the `@es_kwargs` decorator) are explicitly recognized as being claimed by ES and will never be trod upon by future versions of pyelasticsearch. To avoid conflicts, kwargs not yet so recognized should have "es_" prepended by the caller. pyelasticsearch will strip off the "es_" and pass the rest along to ES unscathed. Ideally, we'll then add explicit recognition of those args in a future release.

These "pass-through" kwargs are converted to text as follows:

Bools `True`: "true" `False`: "false"

Strings Passed unmolested

Ints, longs, and floats Converted to strings via `str()`

Lists and tuples Joined with commas, e.g. `['one-index', 'two-index']` becomes `one-index, two-index`

Datetimes and dates Datetimes are converted to ISO strings, like `2001-12-25T13:04:56`, dates convert to midnight: `2001-12-25T00:00:00`.

Anything else raises a TypeError.

API Documentation

A Word About Forward-Compatibility Kwargs

In the following documentation, the phrase *"other kwargs listed below"* refers to the kwargs documented in a subsequent *Parameters* section. However, it also implicitly includes any kwargs the caller might care to make up and have passed to ES as query string parameters. These kwargs must start with `es_` for forward compatibility and will be unprefixed and converted to strings as discussed in *Features*.

ElasticSearch Class

Unless otherwise indicated, methods return the JSON-decoded response sent by elasticsearch. This way, you don't lose any part of the return value, no matter how esoteric. But fear not: if there was an error, an exception will be raised, so it'll be hard to miss.

class `pyelasticsearch.`**`ElasticSearch`**(*urls='http://localhost'*, *timeout=60*, *max_retries=0*, *port=9200*, *username=None*, *password=None*, *ca_certs='/home/docs/checkouts/readthedocs.org/user_builds/pyelasticsearch/envs/l*
packages/certifi/cacert.pem', *client_cert=None*)

An object which manages connections to elasticsearch and acts as a go-between for API calls to it

This object is thread-safe. You can create one instance and share it among all threads.

Parameters

- **`urls`** – A URL or iterable of URLs of ES nodes. These can be full URLs with port numbers, like `http://elasticsearch.example.com:9200`, or you can pass the port separately using the `port` kwarg. To do HTTP basic authentication, you can use RFC-2617-style URLs like `http://someuser:somepassword@example.com:9200` or the separate `username` and `password` kwargs below.
- **`timeout`** – Number of seconds to wait for each request before raising Timeout

- **max_retries** – How many other servers to try, in series, after a request times out or a connection fails
- **username** – Authentication username to send via HTTP basic auth
- **password** – Password to use in HTTP basic auth. If a username and password are embedded in a URL, those are favored.
- **port** – The default port to connect on, for URLs that don't include an explicit port
- **ca_certs** – A path to a bundle of CA certificates to trust. The default is to use Mozilla's bundle, the same one used by Firefox.
- **client_cert** – A certificate to authenticate the client to the server

json_encoder = <class 'pyelasticsearch.client.JsonEncoder'>
You can set this attribute on an instance to customize JSON encoding. The stock JsonEncoder class maps Python datetimes to ES-style datetimes and Python sets to ES lists. You can subclass it to add more.

Bulk Indexing Methods

class pyelasticsearch.**ElasticSearch**

bulk(*actions*, *index=None*, *doc_type=None*[, *other kwargs listed below*])
Perform multiple index, delete, create, or update actions per request.

Used with helper routines *index_op()*, *delete_op()*, and *update_op()*, this provides an efficient, readable way to do large-scale changes. This contrived example illustrates the structure:

```
es.bulk([es.index_op({'title': 'All About Cats', 'pages': 20}),
         es.index_op({'title': 'And Rats', 'pages': 47}),
         es.index_op({'title': 'And Bats', 'pages': 23})],
        doc_type='book',
        index='library')
```

More often, you'll want to index (or delete or update) a larger number of documents. In those cases, yield your documents from a generator, and use *bulk_chunks()* to divide them into multiple requests:

```
from pyelasticsearch import bulk_chunks

def documents():
    for book in books:
        yield es.index_op({'title': book.title, 'pages': book.pages})
        # index_op() also takes kwargs like index= and id= in case
        # you want more control.
        #
        # You could also yield some delete_ops or update_ops here.

# bulk_chunks() breaks your documents into smaller requests for speed:
for chunk in bulk_chunks(documents(),
                         docs_per_chunk=500,
                         bytes_per_chunk=10000):
    # We specify a default index and doc type here so we don't
    # have to repeat them in every operation:
    es.bulk(chunk, doc_type='book', index='library')
```

Parameters

- **actions** – An iterable of bulk actions, generally the output of *bulk_chunks()* but sometimes a list of calls to *index_op()*, *delete_op()*, and *update_op()* directly. Specifically, an iterable of JSON-encoded bytestrings that can be joined with newlines and sent to ES.
- **index** – Default index to operate on
- **doc_type** – Default type of document to operate on. Cannot be specified without `index`.
- **consistency** – See the ES docs.
- **refresh** – See the ES docs.
- **replication** – See the ES docs.
- **routing** – See the ES docs.
- **timeout** – See the ES docs.

Return the decoded JSON response on success.

Raise *BulkError* if any of the individual actions fail. The exception provides enough about the failed actions to identify them for retrying.

Sometimes there is an error with the request in general, not with any individual actions. If there is a connection error, timeout, or other transport error, a more general exception will be raised, as with other methods; see *Error Handling*.

See ES's bulk API for more detail.

index_op(*doc*, *doc_type=None*, *overwrite_existing=True*, ***meta*)

Return a document-indexing operation that can be passed to *bulk()*. (See there for examples.)

Specifically, return a 2-line, JSON-encoded bytestring.

Parameters

- **doc** – A mapping of property names to values.
- **doc_type** – The type of the document to index, if different from the one you pass to *bulk()*
- **overwrite_existing** – Whether we should overwrite existing documents of the same ID and doc type. (If False, this does a *create* operation.)
- **meta** – Other args controlling how the document is indexed, like `id` (most common), `index` (next most common), `version`, and `routing`. See ES's bulk API for details on these.

delete_op(*doc_type=None*, ***meta*)

Return a document-deleting operation that can be passed to *bulk()*.

```
def actions():
    ...
    yield es.delete_op(id=7)
    yield es.delete_op(id=9,
                       index='some-non-default-index',
                       doc_type='some-non-default-type')
    ...

es.bulk(actions(), ...)
```

Specifically, return a JSON-encoded bytestring.

Parameters

- **doc_type** – The type of the document to delete, if different from the one passed to *bulk()*
- **meta** – A description of what document to delete and how to do it. Example: `{"index": "library", "id": 2, "version": 4}`. See ES's bulk API for a list of all the options.

update_op(*doc=None*, *doc_type=None*, *upsert=None*, *doc_as_upsert=None*, *script=None*, *params=None*, *lang=None*, ***meta*)

Return a document-updating operation that can be passed to *bulk()*.

```
def actions():
    ...
    yield es.update_op(doc={'pages': 4},
                       id=7,
                       version=21)
    ...

es.bulk(actions(), ...)
```

Specifically, return a JSON-encoded bytestring.

Parameters

- **doc** – A partial document to be merged into the existing document
- **doc_type** – The type of the document to update, if different from the one passed to *bulk()*
- **upsert** – The content for the new document created if the document does not exist
- **script** – The script to be used to update the document
- **params** – A dict of the params to be put in scope of the script
- **lang** – The language of the script. Omit to use the default, specified by `script.default_lang`.
- **meta** – Other args controlling what document to update and how to do it, like `id`, `index`, and `retry_on_conflict`, destined for the action line itself rather than the payload. See ES's bulk API for details on these.

bulk_index(*index*, *doc_type*, *docs*, *id_field='id'*, *parent_field='_parent'*[, *other kwargs listed below*])

Index a list of documents as efficiently as possible.

Note: This is deprecated in favor of *bulk()*, which supports all types of bulk actions, not just indexing, is compatible with *bulk_chunks()* for batching, and has a simpler, more flexible design.

Parameters

- **index** – The name of the index to which to add the document. Pass None if you will specify indices individual in each doc.
- **doc_type** – The type of the document
- **docs** – An iterable of Python mapping objects, convertible to JSON, representing documents to index

- **id_field** – The field of each document that holds its ID. Removed from document before indexing.
- **parent_field** – The field of each document that holds its parent ID, if any. Removed from document before indexing.
- **index_field** – The field of each document that holds the index to put it into, if different from the `index` arg. Removed from document before indexing.
- **type_field** – The field of each document that holds the doc type it should become, if different from the `doc_type` arg. Removed from the document before indexing.
- **consistency** – See the ES docs.
- **refresh** – See the ES docs.
- **replication** – See the ES docs.
- **routing** – See the ES docs.
- **timeout** – See the ES docs.

Raise *BulkError* if the request as a whole succeeded but some of the individual actions failed. You can pull enough about the failed actions out of the exception to identify them for retrying.

See ES's bulk API for more detail.

There's also a helper function, outside the ElasticSearch class:

pyelasticsearch.**bulk_chunks**(*actions*, *docs_per_chunk=300*, *bytes_per_chunk=None*)

Return groups of bulk-indexing operations to send to *bulk()*.

Return an iterable of chunks, each of which is a JSON-encoded line or pair of lines in the format understood by ES's bulk API.

Parameters

- **actions** – An iterable of bulk actions, JSON-encoded. The best idea is to pass me a list of the outputs from *index_op()*, *delete_op()*, and *update_op()*.
- **docs_per_chunk** – The number of documents (or, more technically, actions) to put in each chunk. Set to None to use only `bytes_per_chunk`.
- **bytes_per_chunk** – The maximum number of bytes of HTTP body payload to put in each chunk. Leave at None to use only `docs_per_chunk`. This option helps prevent timeouts when you have occasional very large documents. Without it, you may get unlucky: several large docs might land in one chunk, and ES might time out.

Chunks are capped by `docs_per_chunk` or `bytes_per_chunk`, whichever is reached first. Obviously, we cannot make a chunk to smaller than its smallest doc, but we do the best we can. If both `docs_per_chunk` and `bytes_per_chunk` are None, all docs end up in one big chunk (and you might as well not use this at all).

Other Methods

class pyelasticsearch.**ElasticSearch**

aliases(*index=None*[, *other kwargs listed below*])

close_index(*index*)

Close an index.

Parameters **index** – The index to close

See ES's close-index API for more detail.

cluster_state(*metric='_all'*, *index='_all'*[, *other kwargs listed below*])
Return state information about the cluster.

Parameters

- **metric** – Which metric to return: one of "version", "master_node", "nodes", "routing_table", "meatadata", or "blocks", an iterable of them, or a comma-delimited string of them. Defaults to all metrics.
- **index** – An index or iterable of indexes to return info about
- **local** – See the ES docs.

See ES's cluster-state API for more detail.

count(*query*[, *other kwargs listed below*])
Execute a query against one or more indices and get hit count.

Parameters

- **query** – A dictionary that will convert to ES's query DSL or a string that will serve as a textual query to be passed as the `q` query string parameter
- **index** – An index or iterable of indexes to search. Omit to search all.
- **doc_type** – A document type or iterable thereof to search. Omit to search all.
- **df** – See the ES docs.
- **analyzer** – See the ES docs.
- **default_operator** – See the ES docs.
- **source** – See the ES docs.
- **routing** – See the ES docs.

See ES's count API for more detail.

create_index(*index*, *settings=None*)
Create an index with optional settings.

Parameters

- **index** – The name of the index to create
- **settings** – A dictionary of settings

If the index already exists, raise *IndexAlreadyExistsError*.

See ES's create-index API for more detail.

delete(*index*, *doc_type*, *id*[, *other kwargs listed below*])
Delete a typed JSON document from a specific index based on its ID.

Parameters

- **index** – The name of the index from which to delete
- **doc_type** – The type of the document to delete
- **id** – The (string or int) ID of the document to delete
- **routing** – See the ES docs.

- **parent** – See the ES docs.
- **replication** – See the ES docs.
- **consistency** – See the ES docs.
- **refresh** – See the ES docs.

See ES's delete API for more detail.

delete_all_indexes()
Delete all indexes.

delete_all(*index*, *doc_type*[, *other kwargs listed below*])
Delete all documents of the given doc type from an index.

Parameters

- **index** – The name of the index from which to delete. ES does not support this being empty or "_all" or a comma-delimited list of index names (in 0.19.9).
- **doc_type** – The name of a document type
- **routing** – See the ES docs.
- **parent** – See the ES docs.
- **replication** – See the ES docs.
- **consistency** – See the ES docs.
- **refresh** – See the ES docs.

See ES's delete API for more detail.

delete_by_query(*index*, *doc_type*, *query*[, *other kwargs listed below*])
Delete typed JSON documents from a specific index based on query.

Parameters

- **index** – An index or iterable thereof from which to delete
- **doc_type** – The type of document or iterable thereof to delete
- **query** – A dictionary that will convert to ES's query DSL or a string that will serve as a textual query to be passed as the `q` query string parameter. (Passing the `q` kwarg yourself is deprecated.)
- **q** – See the ES docs.
- **df** – See the ES docs.
- **analyzer** – See the ES docs.
- **default_operator** – See the ES docs.
- **sourcerouting** – See the ES docs.
- **replication** – See the ES docs.
- **consistency** – See the ES docs.

See ES's delete-by-query API for more detail.

delete_index(*index*)
Delete an index.

Parameters **index** – An index or iterable thereof to delete

If the index is not found, raise `ElasticHttpNotFoundError`.

See ES's delete-index API for more detail.

flush(*index=None*[, *other kwargs listed below*])
Flush one or more indices (clear memory).

Parameters

- **index** – An index or iterable of indexes
- **refresh** – See the ES docs.

See ES's flush API for more detail.

gateway_snapshot(*index=None*)
Gateway snapshot one or more indices.

Parameters **index** – An index or iterable of indexes

See ES's gateway-snapshot API for more detail.

get(*index*, *doc_type*, *id*[, *other kwargs listed below*])
Get a typed JSON document from an index by ID.

Parameters

- **index** – The name of the index from which to retrieve
- **doc_type** – The type of document to get
- **id** – The ID of the document to retrieve
- **realtime** – See the ES docs.
- **fields** – See the ES docs.
- **routing** – See the ES docs.
- **preference** – See the ES docs.
- **refresh** – See the ES docs.

See ES's get API for more detail.

get_mapping(*index=None*, *doc_type=None*)
Fetch the mapping definition for a specific index and type.

Parameters

- **index** – An index or iterable thereof
- **doc_type** – A document type or iterable thereof

Omit both arguments to get mappings for all types and indexes.

See ES's get-mapping API for more detail.

get_settings(*index*[, *other kwargs listed below*])
Get the settings of one or more indexes.

Parameters **index** – An index or iterable of indexes

See ES's get-settings API for more detail.

health(*index=None*[, *other kwargs listed below*])
Report on the health of the cluster or certain indices.

Parameters

- **index** – The index or iterable of indexes to examine
- **level** – See the ES docs.
- **wait_for_status** – See the ES docs.
- **wait_for_relocating_shards** – See the ES docs.
- **wait_for_nodes** – See the ES docs.
- **timeout** – See the ES docs.

See ES's cluster-health API for more detail.

index(*index*, *doc_type*, *doc*, *id=None*, *overwrite_existing=True*[, *other kwargs listed below*])
Put a typed JSON document into a specific index to make it searchable.

Parameters

- **index** – The name of the index to which to add the document
- **doc_type** – The type of the document
- **doc** – A Python mapping object, convertible to JSON, representing the document
- **id** – The ID to give the document. Leave blank to make one up.
- **overwrite_existing** – Whether we should overwrite existing documents of the same ID and doc type
- **routing** – A value hashed to determine which shard this indexing request is routed to
- **parent** – The ID of a parent document, which leads this document to be routed to the same shard as the parent, unless `routing` overrides it.
- **timestamp** – An explicit value for the (typically automatic) timestamp associated with a document, for use with `ttl` and such
- **ttl** – The time until this document is automatically removed from the index. Can be an integral number of milliseconds or a duration like '1d'.
- **percolate** – An indication of which percolator queries, registered against this index, should be checked against the new document: '*' or a query string like 'color:green'
- **consistency** – An indication of how many active shards the contact node should demand to see in order to let the index operation succeed: 'one', 'quorum', or 'all'
- **replication** – Set to 'async' to return from ES before finishing replication.
- **refresh** – Pass True to refresh the index after adding the document.
- **timeout** – A duration to wait for the relevant primary shard to become available, in the event that it isn't: for example, "5m"
- **fields** – See the ES docs.

See ES's index API for more detail.

more_like_this(*index*, *doc_type*, *id*, *fields*, *body=''*[, *other kwargs listed below*])
Execute a "more like this" search query against one or more fields and get back search hits.

Parameters

- **index** – The index to search and where the document for comparison lives
- **doc_type** – The type of document to find others like
- **id** – The ID of the document to find others like

- **mlt_fields** – The list of fields to compare on
- **body** – A dictionary that will convert to ES's query DSL and be passed as the request body
- **search_type** – See the ES docs.
- **search_indices** – See the ES docs.
- **search_types** – See the ES docs.
- **search_scroll** – See the ES docs.
- **search_size** – See the ES docs.
- **search_from** – See the ES docs.
- **like_text** – See the ES docs.
- **percent_terms_to_match** – See the ES docs.
- **min_term_freq** – See the ES docs.
- **max_query_terms** – See the ES docs.
- **stop_words** – See the ES docs.
- **min_doc_freq** – See the ES docs.
- **max_doc_freq** – See the ES docs.
- **min_word_len** – See the ES docs.
- **max_word_len** – See the ES docs.
- **boost_terms** – See the ES docs.
- **boost** – See the ES docs.
- **analyzer** – See the ES docs.

See ES's more-like-this API for more detail.

multi_get(*ids*, *index=None*, *doc_type=None*, *fields=None*[, *other kwargs listed below*])
Get multiple typed JSON documents from ES.

Parameters

- **ids** – An iterable, each element of which can be either an a dict or an id (int or string). IDs are taken to be document IDs. Dicts are passed through the Multi Get API essentially verbatim, except that any missing `_type`, `_index`, or `fields` keys are filled in from the defaults given in the `doc_type`, `index`, and `fields` args.
- **index** – Default index name from which to retrieve
- **doc_type** – Default type of document to get
- **fields** – Default fields to return

See ES's Multi Get API for more detail.

open_index(*index*)
Open an index.

Parameters **index** – The index to open

See ES's open-index API for more detail.

optimize(*index=None*[, *other kwargs listed below*])
Optimize one or more indices.

Parameters

- **index** – An index or iterable of indexes
- **max_num_segments** – See the ES docs.
- **only_expunge_deletes** – See the ES docs.
- **refresh** – See the ES docs.
- **flush** – See the ES docs.
- **wait_for_merge** – See the ES docs.

See ES's optimize API for more detail.

percolate(*index*, *doc_type*, *doc*[, *other kwargs listed below*])
Run a JSON document through the registered percolator queries, and return which ones match.

Parameters

- **index** – The name of the index to which the document pretends to belong
- **doc_type** – The type the document should be treated as if it has
- **doc** – A Python mapping object, convertible to JSON, representing the document
- **routing** – See the ES docs.
- **preference** – See the ES docs.
- **ignore_unavailable** – See the ES docs.
- **percolate_format** – See the ES docs.

Use *index()* to register percolators. See ES's percolate API for more detail.

put_mapping(*index*, *doc_type*, *mapping*[, *other kwargs listed below*])
Register specific mapping definition for a specific type against one or more indices.

Parameters

- **index** – An index or iterable thereof
- **doc_type** – The document type to set the mapping of
- **mapping** – A dict representing the mapping to install. For example, this dict can have top-level keys that are the names of doc types.
- **ignore_conflicts** – See the ES docs.

See ES's put-mapping API for more detail.

refresh(*index=None*)
Refresh one or more indices.

Parameters **index** – An index or iterable of indexes

See ES's refresh API for more detail.

search(*query*[, *other kwargs listed below*])
Execute a search query against one or more indices and get back search hits.

Parameters

- **query** – A dictionary that will convert to ES's query DSL or a string that will serve as a textual query to be passed as the `q` query string parameter
- **index** – An index or iterable of indexes to search. Omit to search all.
- **doc_type** – A document type or iterable thereof to search. Omit to search all.
- **size** – Limit the number of results to `size`. Use with `es_from` to implement paginated searching.
- **routing** – See the ES docs.

See ES's search API for more detail.

send_request(*method*, *path_components*, *body=''*, *query_params=None*)
Send an HTTP request to ES, and return the JSON-decoded response.

This is mostly an internal method, but it also comes in handy if you need to use a brand new ES API that isn't yet explicitly supported by pyelasticsearch, while still taking advantage of our connection pooling and retrying.

Retry the request on different servers if the first one is down and the `max_retries` constructor arg was > 0.

On failure, raise an *ElasticHttpError*, a *ConnectionError*, or a *Timeout*.

Parameters

- **method** – An HTTP method, like "GET"
- **path_components** – An iterable of path components, to be joined by "/"
- **body** – A map of key/value pairs to be sent as the JSON request body. Alternatively, a string to be sent verbatim, without further JSON encoding.
- **query_params** – A map of querystring param names to values or `None`

status(*index=None*[, *other kwargs listed below*])
Retrieve the status of one or more indices

Parameters

- **index** – An index or iterable thereof
- **recovery** – See the ES docs.
- **snapshot** – See the ES docs.

See ES's index-status API for more detail.

update(*index*, *doc_type*, *id*, *script*[, *other kwargs listed below*])
Update an existing document. Raise `TypeError` if `script`, `doc` and `upsert` are all unspecified.

Parameters

- **index** – The name of the index containing the document
- **doc_type** – The type of the document
- **id** – The ID of the document
- **script** – The script to be used to update the document
- **params** – A dict of the params to be put in scope of the script
- **lang** – The language of the script. Omit to use the default, specified by `script.default_lang`.

- **doc** – A partial document to be merged into the existing document
- **upsert** – The content for the new document created if the document does not exist
- **doc_as_upsert** – The provided document will be inserted if the document does not already exist
- **routing** – See the ES docs.
- **parent** – See the ES docs.
- **timeout** – See the ES docs.
- **replication** – See the ES docs.
- **consistency** – See the ES docs.
- **percolate** – See the ES docs.
- **refresh** – See the ES docs.
- **retry_on_conflict** – See the ES docs.
- **fields** – See the ES docs.

See ES's Update API for more detail.

update_aliases (*settings*[, *other kwargs listed below*])
Atomically add, remove, or update aliases in bulk.

Parameters **actions** – A list of the actions to perform

See ES's indices-aliases API.

update_all_settings (*settings*)
Update the settings of all indexes.

Parameters **settings** – A dictionary of settings

See ES's update-settings API for more detail.

update_settings (*index*, *settings*)
Change the settings of one or more indexes.

Parameters

- **index** – An index or iterable of indexes
- **settings** – A dictionary of settings

See ES's update-settings API for more detail.

Error Handling

Any method representing an ES API call can raise one of the following exceptions:

exception pyelasticsearch.exceptions.**ConnectionError**
Exception raised there is a connection error and we are out of retries. (See the `max_retries` argument to `ElasticSearch`.)

exception pyelasticsearch.exceptions.**Timeout**
Exception raised when an HTTP request times out and we are out of retries. (See the `max_retries` argument to `ElasticSearch`.)

exception pyelasticsearch.exceptions.**BulkError**
Exception raised when one or more bulk actions fail

You can extract document IDs from these to retry them.

errors
Return a list of actions that failed, in the format emitted by ES:

```
{"index" : {
    "_index" : "test",
    "_type" : "type1",
    "_id" : "1",
    "status" : 409,
    "error" : "VersionConflictEngineException[[test][2] [type1][1]: version␣
↪conflict, current [3], provided [2]]"
  }
},
{"update" : {
    "_index" : "index1",
    "_type" : "type1",
    "_id" : "1",
    "status" : 404,
    "error" : "DocumentMissingException[[index1][-1] [type1][1]: document␣
↪missing]"
  }
},
...
```

successes
Return a list of actions that succeeded, in the same format as *errors()*.

exception pyelasticsearch.exceptions.**ElasticHttpError**
Exception raised when ES returns a non-OK (>=400) HTTP status code

error
A string error message

status_code
The HTTP status code of the response that precipitated the error

exception pyelasticsearch.exceptions.**ElasticHttpNotFoundError**
Exception raised when a request to ES returns a 404

exception pyelasticsearch.exceptions.**IndexAlreadyExistsError**
Exception raised on an attempt to create an index that already exists

exception pyelasticsearch.exceptions.**InvalidJsonResponseError**
Exception raised in the unlikely event that ES returns a non-JSON response

input
Return the data we attempted to convert to JSON.

Debugging

pyelasticsearch logs to the `elasticsearch.trace` logger using the Python logging module. If you configure that to show INFO-level messages, then it'll show the requests in curl form and their responses. To see when a server is marked as dead, follow the `elasticsearch` logger.

```
import logging

logging.getLogger('elasticsearch.trace').setLevel(logging.INFO)
logging.getLogger('elasticsearch').setLevel(logging.INFO)
```

Note: This assumes that logging is already set up with something like this:

```
import logging

logging.basicConfig()
```

pyelasticsearch will log lines like:

```
INFO:elasticsearch.trace: curl
-XGET 'http://localhost:9200/fooindex/testdoc/_search' -d '{"fa
cets": {"topics": {"terms": {"field": "topics"}}}}'
```

You can copy and paste the curl line, and it'll work on the command line.

Comparison with elasticsearch-py, the "Official Client"

pyelasticsearch was created before Elasticsearch-the-company provided its own client libraries for anything other than Java. There was no reliable, large-scale ES client for Python: pyes was closest, but it suffered from unreliability and pervasive weirdness, like closing sockets in `__del__` and doing things which were obvious no-ops. We adapted pyelasticsearch from an older, very simple client library and gave it a complete API overhaul in version 0.2, inspired by the principles of poetic API design.

Elasticsearch-the-company later created its own clients, with a strong leaning toward keeping them similar across languages for ease of support and maintenance. The upside is that their libraries always support the latest ES features, down to every last nook and cranny, because the relevant parts are autogenerated from a generic API description language. The downside is that they feel autogenerated: some things end up less than Pythonic.

Which Should You Use?

The official Python client borrows much design—and code—from pyelasticsearch. Starting in 1.0, we return the favor, using elasticsearch-py's transport layer rather than maintaining our own. The important differences remain at the API level.

In general, pyelasticsearch focuses on...

- Pythonic-ness

  pyelasticsearch is designed to feel elegant to the caller. For example, we strive for symmetry: creating an index is `es.create()`, and searching one is `es.search()`. In elasticsearch-py, creating an index is nested inside `es.indices.create(<index name>)`, an artifact of code organization. The tradeoff for added design thought is that the project moves slower.

- Good defaults and simple interfaces

  For example, there is only a single transport, HTTP(S), but it is almost always the right one. Thrift, the leading alternative, yields a 15% speed boost but only when using many small requests. It doesn't help at all for bulk indexing, where speed is most often a concern, and it complicates troubleshooting, proxying, and setup. In fact, it's deprecated in ES 1.5 and will be removed in 2.0.

For another example, if you use an HTTPS URL, the authenticity of the server certificate will be automatically verified using Mozilla's certificate authority store. You neither have to manually enable verification nor provide your own store.

The tradeoff here is that we don't expose as many knobs to twiddle as the official client. If you have unusual needs, like using self-signed SSL certificates, we might not be for you. Otherwise, you can enjoy less verbose code.

- Safety

  If something fails, it always raises an exception, making it hard to accidentally ignore. elasticsearch-py doesn't always do this: you need to check for errors explicitly when using its bulk indexing helper, for example.

  In addition, data loss is hard to stumble into; we put up guiderails. For example, calling the update-settings API with no indices would, if we simply followed the ES REST API, update all indices, a far-reaching destructive action caused by an omission. We require the explicit use of an `update_all_settings()` method if you want to do this.

- Better documentation

  You should never need to read the source code to figure out what to do. In order to twiddle many of the aforementioned knobs in elasticsearch-py, you must squirrel kwargs down through multiple undocumented layers, from constructor to constructor, until something finally understands them. On the way, it's often unclear what's public and what's private.

  Our top-level docs are comprehensive with regard to our API, we link to the ES docs for details about their system, and we try to respect the Law of Demeter in our layering.

Conversely, elasticsearch-py focuses on...

- Exhaustive functionality

  It provides explicit hooks into every corner of ES and keeps up to date with ES releases.

  Our strategy is to provide *Forward-Compatibility Kwargs* (which elasticsearch-py adopted as well) and *`send_request()`* for the period between an ES release and when we design APIs for its new features.

- Cross-language homogeneity

  If you're using ES from multiple languages every day, you might enjoy an API that looks similar across them.

  Conversely, we aim for idiomatic Python.

Migrating From `pyes`

Moving your project from `pyes` to `pyelasticsearch` is easy, especially for simple use cases. Here are some code changes that will aid your porting.

- `pyelasticsearch` requires `requests` 1.x. Breaking changes were introduced in `requests` 1.0, so if your project was using a previous version, you may need to update your code. Most likely, you just need to change `response.json` to `response.json()`.
- Instantiating the client should be as simple as changing the invocation...

```
pyes.ES(host, **kwargs)
```

...to...

```
pyelasticsearch.ElasticSearch(host, **kwargs)
```

- `pyelasticsearch` has no method `create_index_if_missing`. Instead, you'll need catch the exception manually:

```
try:
    connection.create_index(index='already_existing_index')
except pyelasticsearch.IndexAlreadyExistsError as ex:
    print 'Index already exists, moving on...'
```

- Instead of using `pyes`'s `_send_request`, use *`send_request()`*. This also requires the path to be passed as an iterable instead of a string. For example...

```
es._send_request('POST', 'my_index/my_doc_type', body)
```

...becomes...

```
connection.send_request('POST', ['my_index', 'my_doc_type'], body)
```

- The `indices` keyword argument in `pyes` turns to `index` in `pyelasticsearch`, whether the method takes multiple indices or not.
- The `doc_types` keyword argument in `pyes` turns to `doc_type` in `pyelasticsearch`.
- *`get()`* will raise *`ElasticHttpNotFoundError`* if the requested documents are not found.
- `pyes` expects arguments to `index` to be in a different order than our *`index()`*. The document to be indexed needs to be moved from the first positional argument to the third.
- *`send_request()`* will raise an error if the response can't be converted to JSON. If you expect that a response will not be JSON, catch the exception and inspect the status code. For example...

```
connection = ElasticSearch(host)
try:
    # Check for the existence of the "pycon" index:
    connection.send_request('HEAD', ['pycon'])
except InvalidJsonResponseError as exc:
    if exc.response.status_code == 200:
        print 'The index exists!'
```

- If using `search_raw` from `pyes`, you can use *`search()`* and, if necessary, rename the keyword arguments.

Changelog

v1.4

- Add support for custom certificate authorities via the `ca_certs` arg to the `ElasticSearch` constructor.
- Add support for client certificates via the `client_cert` arg.

v1.3

- Add support for HTTPS.
- Add username, password, and port kwargs to the constructor so you don't have to repeat their values if they're the same across many servers.

v1.2.4 (2015-05-21)

- Don't crash when the `query_params` kwarg is omitted from calls to `send_request()`.

v1.2.3 (2015-04-17)

- Make `delete_all_indexes()` work.
- Fix a bug in which specifying `_all` as an index name sometimes caused doctype names to be treated as index names.

v1.2.2 (2015-04-10)

- Correct a typo in the `bulk()` docs.

v1.2.1 (2015-04-09)

- Update ES doc links, now that Elastic has changed domains and reorganized its docs.
- Require elasticsearch lib 1.3 or greater, as that's when it started exposing `ConnectionTimeout`.

v1.2 (2015-03-06)

- Make sure the Content-Length header gets set when calling `create_index()` with no explicit `settings` arg. This solves 411s when using nginx as a proxy.
- Add `doc_as_upsert()` arg to `update()`.
- Make `bulk_chunks()` compute perfectly optimal results, no longer ever exceeding the byte limit unless a single document is over the limit on its own.

v1.1 (2015-02-12)

- Introduce new bulk API, supporting all types of bulk operations (index, update, create, and delete), providing chunking via `bulk_chunks()`, and introducing per-action error-handling. All errors raise exceptions–even individual failed operations–and the exceptions expose enough data to identify operations for retrying or reporting. The design is decoupled in case you want to create your own chunkers or operation builders.
- Deprecate `bulk_index()` in favor of the more capable `bulk()`.
- Make one last update to `bulk_index()`. It now catches individual operation failures, raising `BulkError`. Also add the `index_field` and `type_field` args, allowing you to index across different indices and doc types within one request.
- `ElasticSearch` object now defaults to http://localhost:9200/ if you don't provide any node URLs.
- Improve docs: give a better overview on the front page, and document how to customize JSON encoding.

v1.0 (2015-01-23)

- Switch to elasticsearch-py's transport and downtime-pooling machinery, much of which was borrowed from us anyway.
- Make bulk indexing (and likely other network things) 15 times faster.
- Add a comparison with the official client to the docs.
- Fix `delete_by_query()` to work with ES 1.0 and later.
- Bring `percolate()` es_kwargs up to date.
- Fix all tests that were failing on modern versions of ES.
- Tolerate errors that are non-strings and create exceptions for them properly.

Note: Backward incompatible:

- Drop compatibility with elasticsearch < 1.0.
- Redo `cluster_state()` to work with ES 1.0 and later. Arguments have changed.
- InvalidJsonResponseError no longer provides access to the HTTP response (in the `response` property): just the bad data (the `input` property).
- Change from the logger "pyelasticsearch" to "elasticsearch.trace".
- Remove `revival_delay` param from ElasticSearch object.
- Remove `encode_body` param from `send_request()`. Now all dicts are JSON-encoded, and all strings are left alone.

---

v0.7.1 (2014-08-12)

- Brings tests up to date with `update_aliases()` API change.

v0.7 (2014-08-12)

- When an `id_field` is specified for `bulk_index()`, don't index it under its original name as well; use it only as the `_id`.
- Rename `aliases()` to `get_aliases()` for consistency with other methods. Original name still works but is deprecated. Add an `alias` kwarg to the method so you can fetch specific aliases.

Note: Backward incompatible:

- `update_aliases()` no longer requires a dict with an `actions` key; that much is implied. Just pass the value of that key.

---

v0.6.1 (2013-11-01)

- Update package requirements to allow requests 2.0, which is in fact compatible. (Natim)
- Properly raise `IndexAlreadyExistsException` even if the error is reported by a node other than the one to which the client is directly connected. (Jannis Leidel)

v0.6 (2013-07-23)

Note: Note the change in behavior of `bulk_index()` in this release. This change probably brings it more in line with your expectations. But double check, since it now overwrites existing docs in situations where it didn't before.

Also, we made a backward-incompatible spelling change to a little-used `index()` kwarg.

- `bulk_index()` now overwrites any existing doc of the same ID and doctype. Before, in certain versions of ES (like 0.90RC2), it did nothing at all if a document already existed, probably much to your surprise. (We removed the `'op_type': 'create'` pair, whose intentions were always mysterious.) (Gavin Carothers)
- Rename the `force_insert` kwarg of `index()` to `overwrite_existing`. The old name implied the opposite of what it actually did. (Gavin Carothers)

v0.5 (2013-04-20)

- Support multiple indices and doctypes in `delete_by_query()`. Accept both string and JSON queries in the `query` arg, just as `search()` does. Passing the `q` arg explicitly is now deprecated.
- Add `multi_get`.
- Add `percolate`. Thanks, Adam Georgiou and Joseph Rose!
- Add ability to specify the parent document in `bulk_index()`. Thanks, Gavin Carothers!
- Remove the internal, undocumented `from_python` method. django-haystack users will need to upgrade to a newer version that avoids using it.
- Refactor JSON encoding machinery. Now it's clearer how to customize it: just plug your custom JSON encoder class into `ElasticSearch.json_encoder`.
- Don't crash under `python -OO`.
- Support non-ASCII URL path components (like Unicode document IDs) and query string param values.
- Switch to the nose testrunner.

v0.4.1 (2013-03-25)

- Fix a bug introduced in 0.4 wherein "None" was accidentally sent to ES when an ID wasn't passed to `index()`.

v0.4 (2013-03-19)

- Support Python 3.
- Support more APIs:
  - `cluster_state`
  - `get_settings`
  - `update_aliases` and `aliases`
  - `update` (existed but didn't work before)
- Support the `size` param of the `search` method. (You can now change `es_size` to `size` in your code if you like.)

- Support the `fields` param on `index` and `update` methods, new since ES 0.20.
- Maintain better precision of floats when passed to ES.
- Change endpoint of bulk indexing so it works on ES < 0.18.
- Support documents whose ID is 0.
- URL-escape path components, so doc IDs containing funny chars work.
- Add a dedicated `IndexAlreadyExistsError` exception for when you try to create an index that already exists. This helps you trap this situation unambiguously.
- Add docs about upgrading from pyes.
- Remove the undocumented and unused `to_python` method.

v0.3 (2013-01-10)

- Correct the `requests` requirement to require a version that has everything we need. In fact, require requests 1.x, which has a stable API.
- Add `update()` method.
- Make `send_request` method public so you can use ES APIs we don't yet explicitly support.
- Handle JSON translation of Decimal class and sets.
- Make `more_like_this()` take an arbitrary request body so you can filter the returned docs.
- Replace the `fields` arg of `more_like_this` with `mlt_fields`. This makes it actually work, as it's the param name ES expects.
- Make explicit our undeclared dependency on simplejson.

v0.2 (2012-10-06)

Many thanks to Erik Rose for almost completely rewriting the API to follow best practices, improve the API user experience, and make pyelasticsearch future-proof.

Note: This release is **backward-incompatible** in numerous ways, please read the following section carefully. If in doubt, you can easily stick with pyelasticsearch 0.1.

Backward-incompatible changes:

- Simplify `search()` and `count()` calling conventions. Each now supports either a textual or a dict-based query as its first argument. There's no longer a need to, for example, pass an empty string as the first arg in order to use a JSON query (a common case).
- Standardize on the singular for the names of the `index` and `doc_type` kwargs. It's not always obvious whether an ES API allows for multiple indexes. This was leading me to have to look aside to the docs to determine whether the kwarg was called `index` or `indexes`. Using the singular everywhere will result in fewer doc lookups, especially for the common case of a single index.
- Rename `morelikethis` to `more_like_this` for consistency with other methods.
- `index()` now takes `(index, doc_type, doc)` rather than `(doc, index, doc_type)`, for consistency with `bulk_index()` and other methods.

- Similarly, `put_mapping()` now takes `(index, doc_type, mapping)` rather than `(doc_type, mapping, index)`.
- To prevent callers from accidentally destroying large amounts of data...
  - `delete()` no longer deletes all documents of a doctype when no ID is specified; use `delete_all()` instead.
  - `delete_index()` no longer deletes all indexes when none are given; use `delete_all_indexes()` instead.
  - `update_settings()` no longer updates the settings of all indexes when none are specified; use `update_all_settings()` instead.
- `setup_logging()` is gone. If you want to configure logging, use the logging module's usual facilities. We still log to the "pyelasticsearch" named logger.
- Rethink error handling:
  - Raise a more specific exception for HTTP error codes so callers can catch it without examining a string.
  - Catch non-JSON responses properly, and raise the more specific `NonJsonResponseError` instead of the generic `ElasticSearchError`.
  - Remove mentions of nonexistent exception types that would cause crashes in their `except` clauses.
  - Crash harder if JSON encoding fails: that always indicates a bug in pyelasticsearch.
  - Remove the ill-defined `ElasticSearchError`.
  - Raise `ConnectionError` rather than `ElasticSearchError` if we can't connect to a node (and we're out of auto-retries).
  - Raise `ValueError` rather than `ElasticSearchError` if no documents are passed to `bulk_index`.
  - All exceptions are now more introspectable, because they don't immediately mash all the context down into a string. For example, you can recover the unmolested response object from `ElasticHttpError`.
  - Removed `quiet` kwarg, meaning we always expose errors.

Other changes:

- Add Sphinx documentation.
- Add load-balancing across multiple nodes.
- Add failover in the case where a node doesn't respond.
- Add `close_index`, `open_index`, `update_settings`, `health`.
- Support passing arbitrary kwargs through to the ES query string. Known ones are taken verbatim; unanticipated ones need an "es_" prefix to guarantee forward compatibility.
- Automatically convert `datetime` objects when encoding JSON.
- Recognize and convert datetimes and dates in pass-through kwargs. This is useful for `timeout`.
- In routines that can take either one or many indexes, don't require the caller to wrap a single index name in a list.
- Many other internal improvements

v0.1 (2012-08-30)

Initial release based on the work of Robert Eanes and other authors

Development Notes

Testing

To run the tests:

```
% python setup.py test
```

This should automatically install the additional dependencies required for testing if you don't have them.

Documentation

Documentation is located in `docs/` and requires Sphinx to build.

To get the requirements:

```
% pip install Sphinx
```

To build the docs:

```
% cd docs/
% make html
```

Documentation committed and pushed to the main repository is available on ReadTheDocs at http://pyelasticsearch.readthedocs.org/.

Philosophy

pyelasticsearch is intended as a low-level, lossless API to elasticsearch. That is, it generally refrains from adding abstractions that limit flexibility or power. For example, it handles JSON conversion because there is a strict one-to-one mapping between JSON and Python dictionaries: nothing is lost. It converts bad HTTP status codes to exceptions, but you can still access the raw codes and responses by drilling into the exceptions.

Therefore, pyelasticsearch is a good choice for building higher-level APIs upon—ones which make common cases easier but where certain edge cases feel like "coloring outside the lines". One such library is elasticutils. However, pyelasticsearch is also meant to be directly usable by humans: a great deal of care has been taken to keep calls brief, understandable, consistent, and error-resistant and to deal in data structures which are easy to manipulate with Python's built-in routines.

Patches along these lines are always welcome. Thank you for trying pyelasticsearch!

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

R

S

T

U