Elasticsearch Documentation

Release 2.4.1

Honza Král

Mar 30, 2018

Contents

Official low-level client for Elasticsearch. Its goal is to provide common ground for all Elasticsearch-related code in Python; because of this it tries to be opinion-free and very extendable.

For a more high level client library with more limited scope, have a look at elasticsearch-dsl - it is a more pythonic library sitting on top of `elasticsearch-py`.

CHAPTER 1

Compatibility

The library is compatible with all Elasticsearch versions since `0.90.x` but you **have to use a matching major version**:

For **Elasticsearch 2.0** and later, use the major version 2 (`2.x.y`) of the library.

For **Elasticsearch 1.0** and later, use the major version 1 (`1.x.y`) of the library.

For **Elasticsearch 0.90.x**, use a version from `0.4.x` releases of the library.

The recommended way to set your requirements in your *setup.py* or *requirements.txt* is:

```
# Elasticsearch 2.x
elasticsearch>=2.0.0,<3.0.0

# Elasticsearch 1.x
elasticsearch>=1.0.0,<2.0.0

# Elasticsearch 0.90.x
elasticsearch<1.0.0
```

The development is happening on `master` and `1.x` branches, respectively.

CHAPTER 2

Example Usage

```
from datetime import datetime
from elasticsearch import Elasticsearch
es = Elasticsearch()

doc = {
    'author': 'kimchy',
    'text': 'Elasticsearch: cool. bonsai cool.',
    'timestamp': datetime.now(),
}
res = es.index(index="test-index", doc_type='tweet', id=1, body=doc)
print(res['created'])

res = es.get(index="test-index", doc_type='tweet', id=1)
print(res['_source'])

es.indices.refresh(index="test-index")

res = es.search(index="test-index", body={"query": {"match_all": {}}})
print("Got %d Hits:" % res['hits']['total'])
for hit in res['hits']['hits']:
    print("%(timestamp)s %(author)s: %(text)s" % hit["_source"])
```

CHAPTER 3

Features

This client was designed as very thin wrapper around Elasticseach's REST API to allow for maximum flexibility. This means that there are no opinions in this client; it also means that some of the APIs are a little cumbersome to use from Python. We have created some *Helpers* to help with this issue as well as a more high level library (elasticsearch-dsl) on top of this one to provide a more convenient way of working with Elasticsearch.

3.1 Persistent Connections

`elasticsearch-py` uses persistent connections inside of individual connection pools (one per each configured or sniffed node). Out of the box you can choose between two `http` protocol implementations. See *Transport classes* for more information.

The transport layer will create an instance of the selected connection class per node and keep track of the health of individual nodes - if a node becomes unresponsive (throwing exceptions while connecting to it) it's put on a timeout by the `ConnectionPool` class and only returned to the circulation after the timeout is over (or when no live nodes are left). By default nodes are randomized before being passed into the pool and round-robin strategy is used for load balancing.

You can customize this behavior by passing parameters to the *Connection Layer API* (all keyword arguments to the `Elasticsearch` class will be passed through). If what you want to accomplish is not supported you should be able to create a subclass of the relevant component and pass it in as a parameter to be used instead of the default implementation.

3.2 Automatic Retries

If a connection to a node fails due to connection issues (raises `ConnectionError`) it is considered in faulty state. It will be placed on hold for `dead_timeout` seconds and the request will be retried on another node. If a connection fails multiple times in a row the timeout will get progressively larger to avoid hitting a node that's, by all indication, down. If no live connection is available, the connection that has the smallest timeout will be used.

By default retries are not triggered by a timeout (`ConnectionTimeout`), set `retry_on_timeout` to `True` to also retry on timeouts.

3.3 Sniffing

The client can be configured to inspect the cluster state to get a list of nodes upon startup, periodically and/or on failure. See `Transport` parameters for details.

Some example configurations:

```
from elasticsearch import Elasticsearch

# by default we don't sniff, ever
es = Elasticsearch()

# you can specify to sniff on startup to inspect the cluster and load
# balance across all nodes
es = Elasticsearch(["seed1", "seed2"], sniff_on_start=True)

# you can also sniff periodically and/or after failure:
es = Elasticsearch(["seed1", "seed2"],
          sniff_on_start=True,
          sniff_on_connection_fail=True,
          sniffer_timeout=60)
```

3.4 Thread safety

The client is thread safe and can be used in a multi threaded environment. Best practice is to create a single global instance of the client and use it throughout your application. If your application is long-running consider turning on *Sniffing* to make sure the client is up to date on the cluster location.

By default we allow `urllib3` to open up to 10 connections to each node, if your application calls for more paralelism, use the `maxsize` parameter to raise the limit:

```
# allow up to 25 connections to each node
es = Elasticsearch(["host1", "host2"], maxsize=25)
```

Note: Since we use persistent connections throughout the client it means that the client doesn't tolerate `fork` very well. If your application calls for multiple processes make sure you create a fresh client after call to `fork`. Note that Python's `multiprocessing` module uses `fork` to create new processes on POSIX systems.

3.5 SSL and Authentication

You can configure the client to use `SSL` for connecting to your elasticsearch cluster, including certificate verification and http auth:

```
from elasticsearch import Elasticsearch

# you can use RFC-1738 to specify the url
```

```
es = Elasticsearch(['https://user:secret@localhost:443'])

# ... or specify common parameters as kwargs

# use certifi for CA certificates
import certifi

es = Elasticsearch(
    ['localhost', 'otherhost'],
    http_auth=('user', 'secret'),
    port=443,
    use_ssl=True,
    verify_certs=True,
    ca_certs=certifi.where(),
)

# SSL client authentication using client_cert and client_key

es = Elasticsearch(
    ['localhost', 'otherhost'],
    http_auth=('user', 'secret'),
    port=443,
    use_ssl=True,
    verify_certs=True,
    ca_certs='/path/to/cacert.pem',
    client_cert='/path/to/client_cert.pem',
    client_key='/path/to/client_key.pem',
)
```

Warning: By default SSL certificates won't be verified, pass in `verify_certs=True` to make sure your certificates will get verified. The client doesn't ship with any CA certificates; easiest way to obtain the common set is by using the certifi package (as shown above).

See class `Urllib3HttpConnection` for detailed description of the options.

3.6 Logging

`elasticsearch-py` uses the standard logging library from python to define two loggers: `elasticsearch` and `elasticsearch.trace`. `elasticsearch` is used by the client to log standard activity, depending on the log level. `elasticsearch.trace` can be used to log requests to the server in the form of `curl` commands using pretty-printed json that can then be executed from command line. If the trace logger has not been configured already it is set to *propagate=False* so it needs to be activated separately.

CHAPTER 4

Environment considerations

When using the client there are several limitations of your environment that could come into play.

When using an http load balancer you cannot use the *Sniffing* functionality - the cluster would supply the client with IP addresses to directly connect to the cluster, circumventing the load balancer. Depending on your configuration this might be something you don't want or break completely.

In some environments (notably on Google App Engine) your http requests might be restricted so that `GET` requests won't accept body. In that case use the `send_get_body_as` parameter of `Transport` to send all bodies via post:

```
from elasticsearch import Elasticsearch
es = Elasticsearch(send_get_body_as='POST')
```

4.1 Running on AWS with IAM

If you want to use this client with IAM based authentication on AWS you can use the requests-aws4auth package:

```
from elasticsearch import Elasticsearch, RequestsHttpConnection
from requests_aws4auth import AWS4Auth

host = 'YOURHOST.us-east-1.es.amazonaws.com'
awsauth = AWS4Auth(YOUR_ACCESS_KEY, YOUR_SECRET_KEY, REGION, 'es')

es = Elasticsearch(
    hosts=[{'host': host, 'port': 443}],
    http_auth=awsauth,
    use_ssl=True,
    verify_certs=True,
    connection_class=RequestsHttpConnection
)
print(es.info())
```

CHAPTER 5

Contents

5.1 API Documentation

All the API calls map the raw REST api as closely as possible, including the distinction between required and optional arguments to the calls. This means that the code makes distinction between positional and keyword arguments; we, however, recommend that people **use keyword arguments for all calls for consistency and safety**.

Note: for compatibility with the Python ecosystem we use `from_` instead of `from` and `doc_type` instead of `type` as parameter names.

5.1.1 Global options

Some parameters are added by the client itself and can be used in all API calls.

Ignore

An API call is considered successful (and will return a response) if elasticsearch returns a 2XX response. Otherwise an instance of `TransportError` (or a more specific subclass) will be raised. You can see other exception and error states in *Exceptions*. If you do not wish an exception to be raised you can always pass in an `ignore` parameter with either a single status code that should be ignored or a list of them:

```
from elasticsearch import Elasticsearch
es = Elasticsearch()

# ignore 400 cause by IndexAlreadyExistsException when creating an index
es.indices.create(index='test-index', ignore=400)

# ignore 404 and 400
es.indices.delete(index='test-index', ignore=[400, 404])
```

Timeout

Global timeout can be set when constructing the client (see `Connection`'s `timeout` parameter) or on a per-request basis using `request_timeout` (float value in seconds) as part of any API call, this value will get passed to the `perform_request` method of the connection class:

```
# only wait for 1 second, regardless of the client's default
es.cluster.health(wait_for_status='yellow', request_timeout=1)
```

Note: Some API calls also accept a `timeout` parameter that is passed to Elasticsearch server. This timeout is internal and doesn't guarantee that the request will end in the specified time.

Response Filtering

The `filter_path` parameter is used to reduce the response returned by elasticsearch. For example, to only return `_id` and `_type`, do:

```
es.search(index='test-index', filter_path=['hits.hits._id', 'hits.hits._type'])
```

It also supports the `*` wildcard character to match any field or part of a field's name:

```
es.search(index='test-index', filter_path=['hits.hits._*'])
```

5.1.2 Elasticsearch

5.1.3 Indices

5.1.4 Cluster

5.1.5 Nodes

5.1.6 Cat

Snapshot —

5.2 Exceptions

5.3 Connection Layer API

All of the classes responsible for handling the connection to the Elasticsearch cluster. The default subclasses used can be overriden by passing parameters to the `Elasticsearch` class. All of the arguments to the client will be passed on to `Transport`, `ConnectionPool` and `Connection`.

For example if you wanted to use your own implementation of the `ConnectionSelector` class you can just pass in the `selector_class` parameter.

Note: `ConnectionPool` and related options (like `selector_class`) will only be used if more than one connection is defined. Either directly or via the *Sniffing* mechanism.

5.3.1 Transport

5.3.2 Connection Pool

5.3.3 Connection Selector

5.3.4 Urllib3HttpConnection (default connection_class)

5.4 Transport classes

List of transport classes that can be used, simply import your choice and pass it to the constructor of `Elasticsearch` as *connection_class*. Note that the `RequestsHttpConnection` requires `requests` to be installed.

For example to use the `requests`-based connection just import it and use it:

```
from elasticsearch import Elasticsearch, RequestsHttpConnection
es = Elasticsearch(connection_class=RequestsHttpConnection)
```

The default connection class is based on `urllib3` which is more performant and lightweight than the optional `requests`-based class. Only use `RequestsHttpConnection` if you have need of any of `requests` advanced features like custom auth plugins etc.

5.4.1 Connection

5.4.2 Urllib3HttpConnection

5.4.3 RequestsHttpConnection

5.5 Helpers

Collection of simple helper functions that abstract some specifics or the raw API.

5.5.1 Bulk helpers

There are several helpers for the `bulk` API since it's requirement for specific formatting and other considerations can make it cumbersome if used directly.

All bulk helpers accept an instance of `Elasticsearch` class and an iterable `actions` (any iterable, can also be a generator, which is ideal in most cases since it will allow you to index large datasets without the need of loading them into memory).

The items in the `action` iterable should be the documents we wish to index in several formats. The most common one is the same as returned by `search()`, for example:

```
{
    '_index': 'index-name',
    '_type': 'document',
    '_id': 42,
    '_parent': 5,
    '_ttl': '1d',
    '_source': {
        "title": "Hello World!",
        "body": "..."
    }
}
```

Alternatively, if *_source* is not present, it will pop all metadata fields from the doc and use the rest as the document data:

```
{
    "_id": 42,
    "_parent": 5,
    "title": "Hello World!",
    "body": "..."
}
```

The `bulk()` api accepts `index`, `create`, `delete`, and `update` actions. Use the `_op_type` field to specify an action (`_op_type` defaults to `index`):

```
{
    '_op_type': 'delete',
    '_index': 'index-name',
    '_type': 'document',
    '_id': 42,
}
{
    '_op_type': 'update',
    '_index': 'index-name',
    '_type': 'document',
    '_id': 42,
    'doc': {'question': 'The life, universe and everything.'}
}
```

Note: When reading raw json strings from a file, you can also pass them in directly (without decoding to dicts first). In that case, however, you lose the ability to specify anything (index, type, even id) on a per-record basis, all documents will just be sent to elasticsearch to be indexed as-is.

5.5.2 Scan

5.5.3 Reindex

5.6 Changelog

5.6.1 2.5.0 (dev)

`elasticsearch-py 2.x` is now available as `elasticsearch2` from PyPI

5.6.2 2.4.1 (2017-01-03)

- don't warn on empty data returned from server
- Propagate request_timeout to scroll calls

5.6.3 2.4.0 (2016-08-17)

- `ping` now ignores all `TransportError` exceptions and just returns `False`
- expose `scroll_id` on `ScanError`
- increase default size for `scan` helper to 1000

Internal:

- changed `Transport.perform_request` to just return the body, not status as well.

5.6.4 2.3.0 (2016-02-29)

- added `client_key` argument to configure client certificates
- debug logging now includes response body even for failed requests

5.6.5 2.2.0 (2016-01-05)

Due to change in json encoding the client will no longer mask issues with encoding - if you work with non-ascii data in python 2 you must use the `unicode` type or have proper encoding set in your environment.

- adding additional options for ssh - `ssl_assert_hostname` and `ssl_assert_fingerprint` to the default connection class
- fix sniffing

5.6.6 2.1.0 (2015-10-19)

- move multiprocessing import inside parallel bulk for Google App Engine

5.6.7 2.0.0 (2015-10-14)

- Elasticsearch 2.0 compatibility release

5.6.8 1.8.0 (2015-10-14)

- removed thrift and memcached connections, if you wish to continue using those, extract the classes and use them separately.
- added a new, parallel version of the bulk helper using thread pools
- In helpers, removed `bulk_index` as an alias for `bulk`. Use `bulk` instead.

5.6.9 1.7.0 (2015-09-21)

- elasticsearch 2.0 compatibility
- thrift now deprecated, to be removed in future version
- make sure urllib3 always uses keep-alive

5.6.10 1.6.0 (2015-06-10)

- Add `indices.flush_synced` API
- `helpers.reindex` now supports reindexing parent/child documents

5.6.11 1.5.0 (2015-05-18)

- Add support for `query_cache` parameter when searching
- helpers have been made more secure by changing defaults to raise an exception on errors
- removed deprecated options `replication` and the deprecated benchmark api.
- Added `AddonClient` class to allow for extending the client from outside

5.6.12 1.4.0 (2015-02-11)

- Using insecure SSL configuration (`verify_cert=False`) raises a warning
- `reindex` accepts a `query` parameter
- enable `reindex` helper to accept any kwargs for underlying `bulk` and `scan` calls
- when doing an initial sniff (via `sniff_on_start`) ignore special sniff timeout
- option to treat `TransportError` as normal failure in `bulk` helpers
- fixed an issue with sniffing when only a single host was passed in

5.6.13 1.3.0 (2014-12-31)

- Timeout now doesn't trigger a retry by default (can be overriden by setting `retry_on_timeout=True`)
- Introduced new parameter `retry_on_status` (defaulting to `(503, 504, )`) controls which http status code should lead to a retry.
- Implemented url parsing according to RFC-1738
- Added support for proper SSL certificate handling
- Required parameters are now checked for non-empty values
- ConnectionPool now checks if any connections were defined
- DummyConnectionPool introduced when no load balancing is needed (only one connection defined)
- Fixed a race condition in ConnectionPool

5.6.14 1.2.0 (2014-08-03)

Compatibility with newest (1.3) Elasticsearch APIs.

- Filter out master-only nodes when sniffing
- Improved docs and error messages

5.6.15 1.1.1 (2014-07-04)

Bugfix release fixing escaping issues with `request_timeout`.

5.6.16 1.1.0 (2014-07-02)

Compatibility with newest Elasticsearch APIs.

- Test helpers - `ElasticsearchTestCase` and `get_test_client` for use in your tests
- Python 3.2 compatibility
- Use `simplejson` if installed instead of stdlib json library
- Introducing a global `request_timeout` parameter for per-call timeout
- Bug fixes

5.6.17 1.0.0 (2014-02-11)

Elasticsearch 1.0 compatibility. See 0.4.X releases (and 0.4 branch) for code compatible with 0.90 elasticsearch.

- major breaking change - compatible with 1.0 elasticsearch releases only!
- Add an option to change the timeout used for sniff requests (`sniff_timeout`).
- empty responses from the server are now returned as empty strings instead of None
- `get_alias` now has `name` as another optional parameter due to issue #4539 in es repo. Note that the order of params have changed so if you are not using keyword arguments this is a breaking change.

5.6.18 0.4.4 (2013-12-23)

- `helpers.bulk_index` renamed to `helpers.bulk` (alias put in place for backwards compatibility, to be removed in future versions)
- Added `helpers.streaming_bulk` to consume an iterator and yield results per operation
- `helpers.bulk` and `helpers.streaming_bulk` are no longer limitted to just index operations.
- unicode body (for `incices.analyze` for example) is now handled correctly
- changed `perform_request` on `Connection` classes to return headers as well. This is a backwards incompatible change for people who have developed their own connection class.
- changed deserialization mechanics. Users who provided their own serializer that didn't extend `JSONSerializer` need to specify a `mimetype` class attribute.
- minor bug fixes

5.6.19 0.4.3 (2013-10-22)

- Fixes to `helpers.bulk_index`, better error handling
- More benevolent `hosts` argument parsing for `Elasticsearch`
- `requests` no longer required (nor recommended) for install

5.6.20 0.4.2 (2013-10-08)

- `ignore` param acceted by all APIs
- Fixes to `helpers.bulk_index`

5.6.21 0.4.1 (2013-09-24)

Initial release.

CHAPTER 6

License

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

Index

E