pylastica Documentation

Release 0.1.2

Joe Linn

July 23, 2014

Contents

CHAPTER 1

Introduction

Pylastica is python port of Nicolas Ruflin's Elastica PHP Elasticsearch client.

CHAPTER 2

Getting Started

2.1 Installation

2.1.1 Requirements

Pylastica requires python v2.7, the urllib3 package, and the dateutils package. For thrift functionality, the thrift package is required, as well.

2.1.2 Installing

```
$ pip install pylastica
```

The urllib3 and dateutils packages will be installed automatically. If you wish to use the thrift transport client, the thrift module must be installed manually:

```
$ pip install thrift
```

2.1.3 Connecting to Elasticsearch

Assuming one Elasticsearch node running locally on port 9200:

```
import pylastica

pylastica_client = pylastica.Client('localhost', 9200)
```

To connect to multiple nodes:

```
import pylastica

pylastica_client = pylastica.Client(connections=[
    {'host': 'localhost', 'port': 9200},
    {'host': 'localhost', 'port': 9201}
])
```

2.2 Indexing Documents

2.2.1 Creating an Index

```
# instantiate the index object
pylastica_index = pylastica_client.get_index('index_name')

# create the index
pylastica_index.create({
    'number_of_shards': 4,
    'number_of_replicas': 1,
    'analysis': {
        'analyzer': {
            'indexAnalyzer': {
                'type': 'custom',
                'tokenizer': 'standard',
                'filter': ['lowercase', 'mySnowball']
            }
        }
    },
    'filter': {
        'mySnowball': {
            'type': 'snowball',
            'language': 'English'
        }
    }
},
True
)
```

2.2.2 Defining a Mapping

```
# create a DocType object
doc_type = pylastica_index.get_doc_type('type_name')

# define the mapping
mapping = pylastica.doc_type.Mapping()
mapping.doc_type = doc_type
mapping.set_param('index_analyzer', 'indexAnalyzer')

# define boost field
mapping.set_param('_boost', {'name': '_boost', 'null_value': 1.0})

mapping.set_properties({
    'id': {'type': 'integer', 'include_in_all': False},
    'user': {
        'type': 'object',
        'properties': {
            'name': {'type': 'string', 'include_in_all': True},
            'fullName': {'type': 'string', 'include_in_all': True}
        }
    },
    '_boost': {'type': 'float', 'include_in_all': False}
})
```

```
# send the mapping to Elasticsearch
mapping.send()
```

2.2.3 Adding Documents

```
# the id of the dicument
id = 1

# create a document
document = pylastica.Document(id, {
    'id': id,
    'user': {
        'name': 'Neo',
        'fullName': 'Thomas Anderson'
    },
    '_boost': 1.0
})

# index the document
doc_type.add_document(document)
```

Elasticsearch indices are typically configured to refresh automatically. However, if you need the document which you have just indexed to be available immediately, it may be necessary to manually refresh the index:

```
doc_type.index.refresh()
```

2.2.4 Bulk Indexing

```
# define documents
documents = [
    pylastica.Document(1, {...})
    pylastica.Document(2, {...})
]

doc_type.add_documents(documents)
```

2.3 Searching

2.3.1 Querying

Each of the Elasticsearch query types is represented as an object in pylastica. These can be found in pylastica's query module. The following example demonstrates the use of a query string query:

```
# instantiate the query object
pylastica_query_string = pylastica.query.QueryString()

# set the query parameters
pylastica_query_string.set_default_operator('AND')
pylastica_query_string.set_query('thomas anderson')

# perform the search
pylastica_result_set = pylastica_index.search(pylastica_query_string)
```

Pagination of search results can be implemented like so:

```
# create a generic Query object using the StringQuery object
pylastica_query = pylastica.query.Query(pylastica_query_string)

# set pagination parameters
pylastica_query.set_from(50)    # start at the 50th result
pylastica_query.set_limit(25)   # return 25 results

# perform the search, this time using the Query object which wraps the QueryString object
pylastica_result_set = pylastica_index.search(pylastica_query)
```

2.3.2 Retrieving Results

```
# extract the actual results from the result set
pylastica_results = pylastica_result_set.results

# get the total number of results
total_results = pylastica_result_set.get_total_hits()

# iterate over the results
for result in pylastica_results:
    result_data = result.data
    # do something with the data
```

2.3.3 Filtering

As with queries, each Elasticsearch filter type is represented as an object in pylastica's filter module. The following example illustrates combining two term filters using an or filter:

```
# filter for the name "neo"
pylastica_filter_name_neo = pylastica.filter.Term('name', 'neo')

pylastica_filter_name_trinity = pylastica.filter.Term('name', 'trinity')

# filter for either "neo" or "trinity"
pylastica_filter_or = pylastica.filter.BoolOr()
pylastica_filter_or.add_filter(pylastica_filter_name_neo)
pylastica_filter_or.add_filter(pylastica_filter_name_trinity)

# add the filter to the Query object
pylastica_query.set_filter(pylastica_filter_or)
```

2.3.4 Facets

The trend of Elasticsearch query constructs represented as objects continues with pylastica's facet implementation. Here is an example using a terms facet:

```
# define a facet
pylastica_facet = pylastica.facet.Terms('facet_name')    # instantiate the Facet object, giving the fa
pylastica_facet.set_field('name')
pylastica_facet.set_size(10)
pylastica_facet.set_order('reverse_count')
```

```
# add the facet to the Query object
pylastica_query.add_facet(pylastica_facet)
```

The above facet will return the 10 least frequently occurring values in the "name" field, along with the count for each value, in order of ascending frequency. Retrieving facet data works as follows:

```
facets = pylastica_result_set.get_facets()
for facet in facets['facet_name']['terms']:
    term = facet['term']    # the value
    count = facet['count']  # the count
```

CHAPTER 3

Indices and tables

- *genindex*
- *modindex*
- *search*