elasticstack Documentation

Release 0.1.0

Ben Lopatin

January 31, 2014

Contents

Django is the web framework for perfectionists with deadlines.

ElasticSearch is a Lucene based search engine and distributed data store with a JSON interface.

Haystack is the fastest way to map Django project models to a search index and search your site.

elasticstack is a set of ElasticSearch specific helpers for Haystack-based projects.

Contents:

CHAPTER 1

elasticstack

Version 0.1.0

Status alpha

Author Ben Lopatin (http://benlopatin.com)

Configurable indexing and other extras for Haystack (with ElasticSearch biases).

Full documentation is on Read the Docs.

1.1 Requirements

- Django: the features in elasticstack have only been tested on 1.4.x.
- Haystack: ElasticSearch support was only added in Haystack 2.x which is still in development. You'll need to install Haystack from source.
- ElasticSearch: presumably any newish version will do, however the only version tested against so far is 0.19.x

1.2 Features and goals

Some of these features are backend agnostic but most features have ElasticSearch in mind.

For more background see the blog post Stretching Haystack's ElasticSearch Backend.

1.2.1 Configurable index mapping

The search mapping provided by Haystack's ElasticSearch backend includes brief but sensible defaults for nGram analysis. You can add change these settings or add your own mappings by providing a mapping dictionary using *ELASTICSEARCH_INDEX_SETTINGS* in your settings file. This example takes the default mapping and adds a synonym analyzer:

```
ELASTICSEARCH_INDEX_SETTINGS = {
    'settings': {
        "analysis": {
            "analyzer": {
                "synonym_analyzer" : {
                    "type": "custom",
```

```
                    "tokenizer" : "standard",
                    "filter" : ["synonym"]
                },
                "ngram_analyzer": {
                    "type": "custom",
                    "tokenizer": "lowercase",
                    "filter": ["haystack_ngram", "synonym"]
                },
                "edgengram_analyzer": {
                    "type": "custom",
                    "tokenizer": "lowercase",
                    "filter": ["haystack_edgengram"]
                }
            },
            "tokenizer": {
                "haystack_ngram_tokenizer": {
                    "type": "nGram",
                    "min_gram": 3,
                    "max_gram": 15,
                },
                "haystack_edgengram_tokenizer": {
                    "type": "edgeNGram",
                    "min_gram": 2,
                    "max_gram": 15,
                    "side": "front"
                }
            },
            "filter": {
                "haystack_ngram": {
                    "type": "nGram",
                    "min_gram": 3,
                    "max_gram": 15
                },
                "haystack_edgengram": {
                    "type": "edgeNGram",
                    "min_gram": 2,
                    "max_gram": 15
                },
                "synonym" : {
                    "type" : "synonym",
                    "ignore_case": "true",
                    "synonyms_path" : "synonyms.txt"
                }
            }
        }
    }
}
```

The synonym filter is ready for your index, but will go unused yet.

The default analyzer for non-nGram fields in Haystack's ElasticSearch backend is the snowball analyzer. A perfectly good analyzer but not necessarily what you need. It's also language specific (English by default).

Specify your analyzer with *ELASTICSEARCH_DEFAULT_ANALYZER* in your settings file:

```
ELASTICSEARCH_DEFAULT_ANALYZER = 'synonym_analyzer'
```

Now all your analyzed fields, except for nGram fields, will be analyzed using *synonym_analyzer*.

1.2.2 Field based analysis

Even with a new default analyzer you may want to change this on a field by field basis as fits your needs. To do so, use the fields from *elasticstack.fields* to specify your analyzer with the *analyzer* keyword argument:

```
from haystack import indexes
from elasticstack.fields import CharField
from myapp.models import MyContent

class MyContentIndex(indexes.SearchIndex, indexes.Indexable):
    text = CharField(document=True, use_template=True,
            analyzer='synonym_analyzer')

    def get_model(self):
        return MyContent
```

1.2.3 Django CBV style views

Haystacks's class based views predate the inclusion of CBVs into the Django core and so the paradigms are different. This makes it harder to impossible to make use of view mixins.

The bundled *SearchView* and *FacetedSearchView* classes are based on *django.views.generic.edit.FormView* using the *SearchMixin* and *FacetedSearchMixin*, respectively. The *SearchMixin* provides the necessary search related attributes and overloads the form processing methods to execute the search.

The *SearchMixin* adds a few search specific attributes:

- *load_all* - a Boolean value for specifying database lookups
- *queryset* - a default *SearchQuerySet*. Defaults to *EmtpySearchQuerySet*
- *search_field* - the name of the form field used for the query. This is added to allow for views which may have more than one search form. Defaults to *q*.

Note: The *SearchMixin* uses the attribute named *queryset* for the resultant *SearchQuerySet*. Naming this attribute *searchqueryset* would make more sense semantically and hew closer to Haystack's naming convention, however by using the *queryset* attribute shared by other Django view mixins it is relatively easy to combine search functionality with other mixins and views.

1.2.4 Management commands

show_mapping

Make a change and wonder why your results don't look as expected? The management command *show_mapping* will print the current mapping for your defined search index(es). At the least it may show that you've simply forgotten to update your index with new mappings:

```
python manage.py show_mapping
```

By default this will display the *existing_mapping* which shows the index, document type, and document properties.:

```
{
    "haystack": {
        "modelresult": {
            "properties": {
                "is_active": {
```

```
                    "type": "boolean"
                },
                "text": {
                    "type": "string"
                },
                "published": {
                    "type": "date",
                    "format": "dateOptionalTime"
                }
            }
        }
    }
}
```

If you provide the *–detail* flag this will return only the field mappings but including additional details, such as boost levels and field-specific analyzers.:

```
{
    "is_active": {
        "index": "not_analyzed",
        "boost": 1,
        "store": "yes",
        "type": "boolean"
    },
    "text": {
        "index": "analyzed",
        "term_vector": "with_positions_offsets",
        "type": "string",
        "analyzer": "custom_analyzer",
        "boost": 1,
        "store": "yes"
    },
    "pub_date": {
        "index": "analyzed",
        "boost": 1,
        "store": "yes",
        "type": "date"
    }
}
```

show_document

Provided the name of an indexed model and a key it generates and prints the generated document for this object:

```
python manage.py show_document myapp.MyModel 19181
```

The JSON document will be formatted with 'pretty' indenting.

1.3 Stability, docs, and tests

The form, view, and backend functionality in this project is considered stable. Test coverage is not substantial, but is run against Django 1.4 through Django 1.6 on Python 2.6 and Python 2.7, Django 1.5 and Django 1.6 on Python 3.3, and Django 1.6 on PyPy.

1.3.1 Why not add this stuff to Haystack?

This project first aims to solve problems related specifically to working with ElasticSearch. Haystack is 1) backend agnostic (a good thing), 2) needs to support existing codebases, and 3) not my project. Most importantly, adding these features through a separate Django app means providing them without needing to fork Haystack. Hopefully some of the features here, once finalized and tested, will be suitable to add to Haystack.

CHAPTER 2

Installation

2.1 Base installation

Installation is straightforward. With your virtualenv activated, use pip to install:

```
$ pip elasticstack
```

Then add *elasticstack* to your Django project's *INSTALLED_APPS*:

```
INSTALLED_APPS = (
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sites",
    "haystack",
    "elasticstack",
),
```

Adding the app to your *INSTALLED_APPS* is necessary to make the management commands available.

2.2 Haystack connection settings

In order to use the configurable ElasticSearch indexing settings you will need to make sure that you're using the project defined backend. Change this:

```
HAYSTACK_CONNECTIONS = {
    'default': {
        'ENGINE': 'haystack.backends.elasticsearch_backend.ElasticsearchSearchEngine',
        'URL': 'http://127.0.0.1:9200/',
        'INDEX_NAME': 'haystack',
    },
}
```

To this:

```
HAYSTACK_CONNECTIONS = {
    'default': {
        'ENGINE': 'elasticstack.backends.ConfigurableElasticSearchEngine',
        'URL': 'http://127.0.0.1:9200/',
        'INDEX_NAME': 'haystack',
    },
}
```

For a full explanation of why and how to customize your index settings, see the *Configurable index mapping* documentation.

CHAPTER 3

Configurable index mapping

ElasticSearch gives you fine grained control over how your indexed content is analyzed, from choosing between built-in analyzers, choosing options for built-in analyzers, and creating your own from existing tokenizers and filters.

Note: An analyzer is a combination of a tokenizer and one or more text filters. The tokenizer is responsible for breaking apart the text into individual "tokens", which could be words or pieces of words. The filters are responsible for transforming and removing tokens from the indexed content, e.g. making all text lowercase, removing common words, indexing synonyms, etc.

The default ElasticSearch backend in Haystack doesn't expose any of this configuration however. The search mapping provided by this backend maps non-nGram text fields to the snowball analyzer. This is a pretty good default for English, but may not meet your requirements and won't work well for non-English languages.

The elasticstack backend takes advantage of the Haystack backend's structure to make it relatively simple to override and extend the search mapping in your project.

elasticstack lets you manage your index mapping in three ways:

1. Changing the default analyzer
2. Specifying an analyzer for an individual *SearchIndex* field
3. Specifying a complete search mapping including custom analyzers

3.1 Haystack configuration

First, you'll need to ensure that you're using the elasticstack backend, not Haystack's. Your *HAYSTACK_CONNECTIONS* should look something like this, so that the *ENGINE* value for any defined search index is using the elasticstack search engine class.:

```
HAYSTACK_CONNECTIONS = {
    'default': {
        'ENGINE': 'elasticstack.backends.ConfigurableElasticSearchEngine',
        'URL': 'http://127.0.0.1:9200/',
        'INDEX_NAME': 'haystack',
    },
}
```

And of course make sure you've followed the instructions for installing Haystack and your ElasticSearch instance.

Important: All of the options described here depend on this configurable search engine backend.

3.2 Chaning the default analyzer

Haystack will map the *snowball* analyzer to non-nGram text content by default.

You can specify an alternate analyzer using the *ELASTICSEARCH_DEFAULT_ANALYZER* setting in your *settings.py* file:

```
ELASTICSEARCH_DEFAULT_ANALYZER = 'stop'
```

Any field that would have been analyzed with the *snowball* analyzer will now use the stop analyzer.

3.3 Choosing a field-specific analyzer

Even with a new default analyzer you may want to change this on a field by field basis as fits your needs. To do so, use the fields from *elasticstack.fields* to specify your analyzer with the *analyzer* keyword argument:

```
from haystack import indexes
from haystack.fields import CharField as BaseCharField
from elasticstack.fields import CharField
from myapp.models import MyContent

class MyContentIndex(indexes.SearchIndex, indexes.Indexable):
    text = CharField(document=True, use_template=True,
            analyzer='stop')
    body = BaseCharField(use_template=True)

    def get_model(self):
        return MyContent
```

Now the *text* field will be indexed using the *stop* analyzer, and the *body* field will be indexed using the default analyzer.

3.4 Custom analyzers and additional configuration

If instead you need to configure an analyzer, define your own, or in any way further customize the search mapping, you can customize the base analysis settings for your index.

You do this by creating a dictionary of analysis settings in your *settings.py* file for the *ELASTICSEARCH_INDEX_SETTINGS* setting.

This example takes the default mapping and adds a synonym analyzer.

```
1  ELASTICSEARCH_INDEX_SETTINGS = {
2      'settings': {
3          "analysis": {
4              "analyzer": {
5                  "synonym_analyzer" : {
6                      "type": "custom",
7                      "tokenizer" : "standard",
8                      "filter" : ["synonym"]
9                  },
10                 "ngram_analyzer": {
11                     "type": "custom",
12                     "tokenizer": "lowercase",
13                     "filter": ["haystack_ngram", "synonym"]
14                 },
```

```
15                      "edgengram_analyzer": {
16                          "type": "custom",
17                          "tokenizer": "lowercase",
18                          "filter": ["haystack_edgengram"]
19                      }
20                  },
21                  "tokenizer": {
22                      "haystack_ngram_tokenizer": {
23                          "type": "nGram",
24                          "min_gram": 3,
25                          "max_gram": 15,
26                      },
27                      "haystack_edgengram_tokenizer": {
28                          "type": "edgeNGram",
29                          "min_gram": 2,
30                          "max_gram": 15,
31                          "side": "front"
32                      }
33                  },
34                  "filter": {
35                      "haystack_ngram": {
36                          "type": "nGram",
37                          "min_gram": 3,
38                          "max_gram": 15
39                      },
40                      "haystack_edgengram": {
41                          "type": "edgeNGram",
42                          "min_gram": 2,
43                          "max_gram": 15
44                      },
45                      "synonym" : {
46                          "type" : "synonym",
47                          "ignore_case": "true",
48                          "synonyms_path" : "synonyms.txt"
49                      }
50                  }
51              }
52          }
53      }
```

The two additions to this mapping are the *synonym_analyzer* at line 5 and the *synonym* filter at line 45.

Adding this mapping in and of itself does nothing more than make your new analyzer available. To use it you either need to change your *ELASTICSEARCH_DEFAULT_ANALYZER* or specify the analyzer in the search index field.

3.5 Realizing custom changes

Even with all of these changes you won't notice any difference in your queries until you've reindexed your content. The mappings for your search index define how that content is handled when it goes into the index; it does nothing for content already there.

CHAPTER 4

Django CBV style views

Haystacks's class based views predate the inclusion of CBVs into the Django core and so the paradigms are different. This makes it harder to impossible to make use of view mixins.

The bundled *SearchView* and *FacetedSearchView* classes are based on *django.views.generic.edit.FormView* using the *SearchMixin* and *FacetedSearchMixin*, respectively. The *SearchMixin* provides the necessary search related attributes and overloads the form processing methods to execute the search.

The *SearchMixin* adds a few search specific attributes:

- *load_all* - a Boolean value for specifying database lookups
- *queryset* - a default *SearchQuerySet*. Defaults to *EmtpySearchQuerySet*
- *search_field* - the name of the form field used for the query. This is added to allow for views which may have more than one search form. Defaults to *q*.

Note: The *SearchMixin* uses the attribute named *queryset* for the resultant *SearchQuerySet*. Naming this attribute *searchqueryset* would make more sense semantically and hew closer to Haystack's naming convention, however by using the *queryset* attribute shared by other Django view mixins it is relatively easy to combine search functionality with other mixins and views.

CHAPTER 5

Management commands

The extra management commands are small tools to help in diagnosing problems with unexpected search results, by showing you how your data is actually mapped for ElasticSearch and how a specific model instance (with a matching *SearchIndex* class) is mapped as an example.

5.1 show_mapping

Make a change and wonder why your results don't look as expected? The management command *show_mapping* will print the current mapping for your defined search index(es). At the least it may show that you've simply forgotten to update your index with new mappings:

```
python manage.py show_mapping
```

By default this will display the *existing_mapping* which shows the index, document type, and document properties.:

```
{
    "haystack": {
        "modelresult": {
            "properties": {
                "is_active": {
                    "type": "boolean"
                },
                "text": {
                    "type": "string"
                },
                "published": {
                    "type": "date",
                    "format": "dateOptionalTime"
                }
            }
        }
    }
}
```

If you provide the *–detail* flag this will return only the field mappings but including additional details, such as boost levels and field-specific analyzers.:

```
{
    "is_active": {
        "index": "not_analyzed",
        "boost": 1,
        "store": "yes",
```

```
        "type": "boolean"
    },
    "text": {
        "index": "analyzed",
        "term_vector": "with_positions_offsets",
        "type": "string",
        "analyzer": "custom_analyzer",
        "boost": 1,
        "store": "yes"
    },
    "pub_date": {
        "index": "analyzed",
        "boost": 1,
        "store": "yes",
        "type": "date"
    }
}
```

5.2 show_document

Provided the name of an indexed model and a key it generates and prints the generated document for this object:

```
python manage.py show_document myapp.MyModel 19181
```

The JSON document will be formatted with 'pretty' indenting.

CHAPTER 6

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

6.1 Types of Contributions

6.1.1 Report Bugs

Report bugs at https://github.com/bennylope/elasticstack/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

6.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

6.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

6.1.4 Write Documentation

elasticstack could always use more documentation, whether as part of the official elasticstack docs, in docstrings, or even on the web in blog posts, articles, and such.

6.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/bennylope/elasticstack/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

6.2 Get Started!

Ready to contribute? Here's how to set up *elasticstack* for local development.

1. Fork the *elasticstack* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/elasticstack.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv elasticstack
   $ cd elasticstack/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 elasticstack tests
$ python setup.py test
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

6.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/bennylope/elasticstack/pull_requests and make sure that the tests pass for all supported Python versions.

6.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_elasticstack
```

CHAPTER 7

Credits

7.1 Development Lead

- Ben Lopatin @bennylope

7.2 Contributors

- Basil Shubin @bashu

CHAPTER 8

History

8.1 0.0.6 (2013-10-04)

- Require pyelasticsearch for installation

8.2 0.0.5 (2013-09-28)

- Fixed reference to old method

8.3 0.0.4 (2013-09-28)

- Search form can search using specified field name
- Added management command to output mapping for an individual document

8.4 0.0.3 (2013-09-28)

- Added default analyzer setting

8.5 0.0.2 (2013-09-28)

- Packaging fix

8.6 0.0.1 (2013-09-28)

- Initial release