Django Stretch Documentation

Release 0.8.0

Alec Koumjian, CBRE Dev

Jan 22, 2019

Contents

The `StretchIndex` class connects a Django model and an Elasticsearch index. Together with the management commands you

CHAPTER 1

Creating an Index

The minimum settings for an index are at least one field and to specify which model it is associated with.

```
from stretch.indices import StretchIndex
from elasticsearch_dsl import Text
from yourapp.models import Foo


class FooIndex(StretchIndex):
    """
    Index the Foo model in Elasticsearch
    """
    name = Text(source='name')

    class Meta:
        model = Foo
```

CHAPTER 2

Tracking Updates

Stretch uses signal handlers to update Elasticsearch documents when a model instance is saved or deleted. There are no signal handlers set up by default, but adding one is really easy.

There are two signal handlers available by default in Stretch: the `RealtimeSignalHandler` which will update your documents synchronously in Elasticsearch and `CelerySignalHandler` which will asynchronously update the documents (you must install and configure celery separately).

Warning: Queryset `update` methods do not trigger the save signal and will not be tracked by default.

```
...
from stretch.signals import RealtimeSignalHandler

class FooIndex(StretchIndex):
    ...

    class Meta:
        ...
        signal_handler = RealtimeSignalHandler
```

CHAPTER 3

Fields

Stretch uses the field classes from Elasticsearch DSL library and accepts all the same arguments, with the addition of the `source` keyword. The `source` keyword should be either a model attribute or a method on the `StretchIndex` class which is used to populate the Elasticsearch field from the model.

```
from elasticsearch_dsl import Text

class FooIndex(StretchIndex):
    name = String(source='name')
    custom_field = String(source='get_custom_value')

    def get_custom_value(self, obj):
        return obj.first_name + obj.last_name
```

CHAPTER 4

Django Rest Framework (DRF) Search Filter

Stretch provides a DRF FilterBackend so that you can use Elasticsearch for your API endpoints. After you have a `StretchIndex` set up for your related model, you can add it to a DRF Viewset.

```
from stretch.drf import StretchSearchFilter


class FooViewSet(ModelViewSet):
    queryset = Foo.objects.all()
    serializer_class = FooSerializer

    # Use dot syntax to point to your StretchIndex class
    stretch_index = 'example.stretch_indices.FooIndex'

    filter_backends = (
        ...
        # Include the search filter in your ``filter_backends``
        StretchSearchFilter,
        ...
    )
```

4.1 Making a Request

When you call your API endpoint, use your DRF *SEARCH_PARAM* to pass your query. By default the value is `search`. If you want to use a different query parameter, subclass `StretchSearchFilter` and set `search_param = [your custom value]`.

```
curl http://localhost:8000/api/foo?search=[your query]
```

4.2 How it Works

DRF Filters implement a single method `filter_queryset` just like a normal DRF filter. Stretch takes the incoming queryset and uses it to filter an Elasticsearch query using object IDs. The filter has a

4.3 Custom Searches

The DRF filter has a default search that includes fuzzy matching and phrase matching. You can set the *default_search_fields* on the index to specify which fields to use. By default is searches all top level fields on the index.

```
class FooIndex(StretchIndex):
    ...

    class Meta:
        ...
        default_search_fields = [
            'field_1',
            'field_3',
            'field_4.autocomplete', # You can use Elasticsearch subfields that use␣
↪different analyzers!
        ]
```

There are two ways to customize the search query. The simplest way is to add a `stretch_modify_search` method on your DRF view. The filter will automatically pass the Elasticsearch DSL object to that method before executing the search. You can modify the search object or create a new one and return it.

```
class FooViewSet(ModelViewSet):
    ...
    stretch_index = 'example.stretch_indices.FooIndex'
    filter_backends = (
        ...
        StretchSearchFilter,
        ...
    )

    def stretch_modify_search(self, s, view, index, request, queryset):
        s = s.filter('terms', tags=['red', 'fish', 'blue'])
        return s
```

For more control you can subclass `StretchSearchFilter`. The only real rule here is to make sure the `filter_queryset` method returns a queryset.

```
from stretch.drf import StretchSearchFilter


class CustomSearchFilter(StretchSearchFilter):
    def filter_queryset(self, request, queryset, view):
        """
        Return a deduplicated list of Foo objects by name
        """
        index = self._get_index(view)
        s = self.build_search(view, index, request, queryset)
        search_results = s.execute()
```

(continues on next page)

(continued from previous page)

```
        # Get unique list of Foo object names
        ordered_names = []
        for bucket in search_results.aggregations.names.buckets:
            ordered_names.append(bucket.key)

        # Filter and order queryset by names
        queryset = queryset.filter(name__in=ordered_names)
        distinct_name_pks = queryset.order_by('name').distinct('name').values_list('pk
↪', flat=True)

        preserved = Case(
            *[When(name=name, then=pos) for pos, name in enumerate(ordered_names)]
        )
        queryset = Foo.objects.filter(pk__in=distinct_name_pks).order_by(preserved)

        return queryset

    def build_search(self, view, index, request, queryset):
        """
        Use aggregations to get to matching unique Foo names
        """
        s = super().build_search(view, index, request, queryset)

        # Retrieve a list of Foo object names
        s.aggs.bucket(
            'names',
            'terms',
            field='name',
            order={'name_score': 'desc'}
        ).bucket(
            'name_score',
            'max',
            script='_score'
        )
        return s
```

CHAPTER 5

Django Stretch

Elasticsearch for Django, Made Friendly

Stretch lets you index and search Django models in Elasticsearch. It sits on top of the `elasticsearch_dsl` package for easy access to most of Elasticsearch's features.

See the complete Documentation

5.1 Installation

Currently you can only install django-stretch via git.

```
pip install git+git@github.com:cbredev/django-stretch.git
```

5.2 Quickstart

1. Make sure you have Elasticsearch installed
2. **Add to `INSTALLED_APPS`**

```
INSTALLED_APPS = (
    ...
    'stretch',
)
```

3. **Create a `stretch_indices.py` file in one of your apps**

```
from stretch.indices import StretchIndex
from stretch.signals import RealtimeSignalHandler
from elasticsearch_dsl import String
from yourapp.models import Foo
```

(continues on next page)

(continued from previous page)

```
class FooIndex(StretchIndex):
    name = String(source='name')
    bar = String(source='get_bar')

    class Meta:
        # Specify the model to track with this index
        model = Foo
        index = 'foos'      # (Optional) specify a custom index name
        doc_type = 'deal'   # (Optional) specify a custom doc type

        # (Optional) Specify any custom index settings
        index_settings = {
            'number_of_replicas': 1
        }
        # Automatically track model updates in realtime or with celery
        signal_handler = RealtimeSignalHandler

    # The bar field is populated by this custom method
    def get_bar(self, obj):
        return 'custom value'
```

4. **Create the index & add all the documents**

```
./manage.py stretch_update_all
```

Your models are now being indexed in Elasticsearch!

5.3 Features

- Specify custom analyzers and other field attributes
- Dynamically update mapping and per index settings
- Django Rest Framework (DRF) Search Filter
- Management Commands to update indices and documents
- post_save / post_delete signals

5.4 Philosophy

Stretch sits on top of the Elasticsearch DSL python library. This makes it really easy to create document mappings and use custom analyzers in a declarative, pythonic syntax. It also includes an up to date interface for performing complex searches. This tool focuses on making it easy to manage your Elasticsearch indices over the long term as you add new fields and analyzers.

5.5 Why not [Haystack|Bungiesearch]?

Both Haystack and Bungieserach are great tools. Haystack works great as a general search engine wrapper, but makes it difficult to use the majority of Elasticsearch's built in features. Bungiesearch does a slightly better job at this, but does not provide good tooling for adding new analyzers or updating index settings. We wanted a tool that let us easily manage indices as well as documents.

5.6 Settings

```
STRETCH_SETTINGS = {
    'HOSTS': ['localhost'],
    'APPS': None,               # If not specified, searches all apps for indices
    'EXCLUDED_INDICES': None,   # Manually ignore an index. You can also set the␣
↪index attribute to `IS_INDEX = False`
    'SIGNAL_PROCESSOR': None,   # Set a global default signal processor. None by␣
↪default
    'TEST': False               # Uses a mock backend so you can run tests in your␣
↪project without trying to make a real connection to Elasticsearch
    'RAISE_EXCEPTIONS': False   # By default, Stretch will use a `logging.exception`␣
↪to notify your application of failures without crashing your threads. If you prefer␣
↪to have your application fail on write errors, set this to `True`
}
```