Django Elastic App Search Documentation

Release 2.0.0

Infoxchange

Oct 02, 2022

Contents

Contents:

CHAPTER 1

Django Elastic App Search

Integrate your Django Project with Elastic App Search with ease.

1.1 Documentation

The full documentation is at https://django_elastic_appsearch.readthedocs.io. Read our step-by-step guide on integrating App Search with your existing Django project over at Medium.

1.2 Dependencies

- Python >= 3.6
- Django >= 2.2
- elastic-enterprise-search
- serpy

1.3 Usage

1.3.1 Installing

Install Django Elastic App Search:

```
pip install django_elastic_appsearch
```

Add it to your *INSTALLED_APPS*:

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

Add the Elastic App Search URL and Key to your settings module:

```
APPSEARCH_HOST = 'localhost:3002'
APPSEARCH_API_KEY = 'some_appsearch_api_token'
```

1.3.2 Configuring app search indexable models

1.3.3 Single engine

Configure the Django models you want to index to Elastic App Search. To index to one engine you can do this by inheriting from the `AppSearchModel`, and then setting some meta options.

`AppsearchMeta.appsearch_engine_name` - Defines which engine in your app search instance your model will be indexed to.

`AppsearchMeta.appsearch_serialiser_class` - Defines how your model object will be serialised when sent to your elastic app search instance. The serialiser and fields used here derives from Serpy, and you can use any of the serpy features like method fields.

Example:

```
from django_elastic_appsearch.orm import AppSearchModel
from django_elastic_appsearch import serialisers

class CarSerialiser(serialisers.AppSearchSerialiser):
    full_name = serialisers.MethodField()
    make = serialisers.StrField()
    model = serialisers.StrField()
    manufactured_year = serialisers.Field()

    def get_full_name(self, instance):
        return '{} {}'.format(make, model)


class Car(AppSearchModel):

    class AppsearchMeta:
        appsearch_engine_name = 'cars'
        appsearch_serialiser_class = CarSerialiser
```

(continues on next page)

(continued from previous page)

```
    make = models.CharField(max_length=100)
    model = models.CharField(max_length=100)
    manufactured_year = models.CharField(max_length=4)
```

1.3.4 Multi engine

Configure the Django models you want to index to Elastic App Search. To index to multiple engines you can do this by inheriting from the `AppSearchMultiEngineModel`, and then setting a meta option.

`AppsearchMeta.appsearch_serialiser_engine_pairs` - A list of tuples of serialisers then engines that define which engine in your app search instance your model will be indexed to and how your model object will be serialised when sent to your elastic app search instance. The serialiser and fields used here derives from Serpy, and you can use any of the serpy features like method fields.

Example:

```
from django_elastic_appsearch.orm import AppSearchModel
from django_elastic_appsearch import serialisers

class CarSerialiser(serialisers.AppSearchSerialiser):
    full_name = serialisers.MethodField()
    make = serialisers.StrField()
    model = serialisers.StrField()
    manufactured_year = serialisers.Field()

    def get_full_name(self, instance):
        return '{} {}'.format(make, model)


class Truck(AppSearchMultiEngineModel):
    """A truck."""

    class AppsearchMeta:
        appsearch_serialiser_engine_pairs = [(CarSerialiser, "trucks")]

    make = models.TextField()
    model = models.TextField()
    year_manufactured = models.DateTimeField()
```

1.3.5 Using model and queryset methods to index and delete documents

Then you can call `index_to_appsearch` and `delete_from_appsearch` from your model objects.

Send the car with id 25 to app search.

```
from mymodels import Car

car = Car.objects.get(id=25)
car.index_to_appsearch()
```

Delete the car with id 21 from app search.

```
from mymodels import Car

car = Car.objects.get(id=21)
car.delete_from_appsearch()
```

Calling these on an `AppSearchModel` will return a single response object, and calling them on an `AppSearchMultiEngineModel` will return a list of response objects.

You can also call `index_to_appsearch` and `delete_from_appsearch` on QuerySets of `AppSearchModel`

Send all cars where the make is 'Toyota' to app search.

```
cars = Car.objects.filter(make='Toyota')
cars.index_to_appsearch()
```

Delete all cars where the make is 'Saab' from app search

```
cars = Car.objects.filter(make='Saab')
cars.delete_from_appsearch()
```

`index_to_appsearch` methods on the QuerySet and your model also supports an optional `update_only` parameter which takes in a boolean value. If `update_only` is set to `True`, the operation on the app search instance will be carried out as a `PATCH` operation. This will be useful if your Django application is only doing partial updates to the documents.

This will also mean that your serialisers can contain a subset of the fields for a document. This will be useful when two or more Django models or applications are using the same app search engine to update different sets of fields on a single document type.

Example below (Continued from the above `Car` example):

```
from django.db import models
from django_elastic_appsearch.orm import AppSearchModel
from django_elastic_appsearch import serialisers

class CarVINNumberSerialiser(serialisers.AppSearchSerialiser):
    vin_number = serialisers.StrField()

class CarVINNumber(AppSearchModel):

    class AppsearchMeta:
        appsearch_engine_name = 'cars'
        appsearch_serialiser_class = CarVINNumberSerialiser

    car = models.OneToOneField(
        Car,
        on_delete=models.CASCADE,
        primary_key=True
    )
    vin_number = models.CharField(max_length=100)

    def get_appsearch_document_id(self):
        return 'Car_{}'.format(self.car.id)
```

```
from mymodels import CarVINNumber

car_vin = CarVINNumber.objects.filter('car__id'=25).first()
```

(continues on next page)

(continued from previous page)

```
car_vin.vin_number = '1M8GDM9A_KP042788'
car_vin.save()
car_vin.refresh_from_db()
car_vin.index_to_appsearch(update_only=True)
```

You'll notice that we've set the `appsearch_engine_name` to `cars` so that the VIN number updates will go through to the same engine. You'll also notice that we've overridden the `get_appsearch_document_id` method to make sure that VIN number updates do go through the same related car document.

The above example will update the car document with id 25 with the new VIN number and leave the data for the rest of the fields intact.

Important note: `PATCH` operations on Elastic App Search cannot create new schema fields if you submit schema fields currently unknown to your engine. So always make sure you're submitting values for existing schema fields on your engine.

1.3.6 Use with your own custom queryset managers

If you want to specify custom managers which also has this functionality, you can inherit from `django_elastic_appsearch.orm.AppSearchQuerySet`

```
from django_elastic_appsearch.orm import AppSearchModel, AppSearchQuerySet

class MyCustomQuerySetManager(AppSearchQuerySet):
    def my_custom_queryset_feature(self):
        # Do Something cool
        pass

class MyCustomModel(AppSearchModel):
    field_1 = models.CharField(max_length=100)

    # Set the custom manager
    objects = MyCustomQuerySetManager.as_manager()
```

1.3.7 Use a custom document id for appsearch

By default, the unique document ID which identifies your model objects in app search is set to `<model_name>_<object_id>`. If we take the car example above, a `Car` object with an id of `543` will have the document ID `Car_543` in app search.

You can customise this value by overriding the `get_appsearch_document_id` method on your model class.

Eg. You can do the following to make sure that the document ID on appsearch is exactly the same as the ID on your model object.

```
class Car(AppSearchModel):

    class AppsearchMeta:
        appsearch_engine_name = 'cars'
        appsearch_serialiser_class = CarSerialiser

    make = models.CharField(max_length=100)
    model = models.CharField(max_length=100)
    manufactured_year = models.CharField(max_length=4)
```

(continues on next page)

(continued from previous page)

```
    def get_appsearch_document_id(self):
        return self.id
```

1.3.8 Settings

This package provides various Django settings entries you can use to configure your connection to the Elastic App Search instance you're using.

APPSEARCH_HOST

- Required: Yes
- Default: No default value

This is a **required** setting to tell your Django application which Elastic App Search instance to connect with.

```
APPSEARCH_HOST = 'localhost:3002'
```

APPSEARCH_API_KEY

- Required: Yes
- Default: No default value

This is a **required** setting to tell your Django application the private key to use to talk to your Elastic App Search instance.

```
APPSEARCH_API_KEY = 'private-key'
```

APPSEARCH_USE_HTTPS

- Required: No
- Default: `True`

This is an **optional** setting to configure whether to use HTTPS or not when your Django application communicates with your Elastic App Search instances. It defaults to `True` if it's not set. This might be useful when you're running your Django project against a local Elastic App Search instance. It's insecure to have this as `False` in a production environment, so make sure to change to `True` in your production version.

```
APPSEARCH_USE_HTTPS = False
```

APPSEARCH_CHUNK_SIZE

- Required: No
- Default: `100`

This is an **optional** setting to configure the chunk size when doing queryset indexing/deleting. Elastic App Search supports upto a 100 documents in one index/destroy request. With this setting, you can change it to your liking. It defaults to the maximum of `100` when this is not set. This might be useful when you want to reduce the size of a request to your Elastic App Search instance when your documents have a lot of fields/data.

```
APPSEARCH_CHUNK_SIZE = 50
```

APPSEARCH_INDEXING_ENABLED

- Required: No
- Default: `True`

This is an **optional** setting to configure if you want to disable indexing to your Elastic App Search instance. This is useful when you want to disable indexing without changing any code. When it's set to `False`, any code where you use `index_to_appsearch()` or `delete_from_appsearch()` will not do anything. It's set to `True` by default when it's not set.

```
APPSEARCH_INDEXING_ENABLED = True
```

Example with all settings entries

```
APPSEARCH_HOST = 'localhost:3002'
APPSEARCH_API_KEY = 'private-key'
APPSEARCH_USE_HTTPS = False
APPSEARCH_CHUNK_SIZE = 50
APPSEARCH_INDEXING_ENABLED = True
```

1.3.9 Writing Tests

This package provides a test case mixin called `MockedAppSearchTestCase` which makes it easier for you to write test cases against `AppSearchModel`'s and `AppSearchMultiEngineModel`'s without actually having to run an Elastic App Search instance during tests.

All you have to do is inherit the mixin, and all the calls to Elastic App Search will be mocked. Example below.

```
from django.test import TestCase
from django_elastic_appsearch.test import MockedAppSearchTestCase
from myapp.test.factories import CarFactory

class BookTestCase(MockedAppSearchTestCase, TestCase):
    def test_indexing_book(self):
        car = CarFactory()
        car.save()
        car.index_to_appsearch()

        self.assertAppSearchModelIndexCallCount(1)
```

You will have access to the following methods to check call counts to different mocked app search methods.

`self.assertAppSearchQuerySetIndexCallCount` — Check the number of times index_to_appsearch was called on a appsearch model querysets.

`self.assertAppSearchQuerySetDeleteCallCount` — Check the number of times delete_from_appsearch was called on an appsearch model querysets.

`self.assertAppSearchModelIndexCallCount` — Check the number of times index_to_appsearch was called on an appsearch model objects.

`self.assertAppSearchModelDeleteCallCount` — Check the number of times delete_from_appsearch was called on an appsearch model objects.

If you are using a subclass of *AppSearchQuerySet* that overrides methods without calling the super class version you can use the *queryset_class* key word argument to the *setUp* function to mock it. Example below.

```
from django.test import TestCase
from django_elastic_appsearch.test import MockedAppSearchTestCase

class BusTestCase(MockedAppSearchTestCase, TestCase):
    """Test the `MockedAppSearchTestCase`."""

    def setUp(self, *args, **kwargs):
        """Load test data."""
        kwargs['queryset_class'] = 'example.querysets.CustomQuerySet.'
        super().setUp(*args, **kwargs)
```

1.3.10 Using the elastic app search python client

We use the official elastic app search python client under the hood to communicate with the app search instance. So if needed, you can access the app search instance directly and use the functionality of the official elastic app search client. Example below.

```
from django_elastic_appsearch.clients import get_api_v1_enterprise_search_client

client = get_api_v1_enterprise_search_client()
client.search('cars', 'Toyota Corolla', {})
```

1.4 Contributing

Contributors are welcome!

- Prior to opening a pull request, please create an issue to discuss the change/feature you've written/thinking of writing if it doesn't already exist.
- Please write simple code and concise documentation, when appropriate.
- Please write test cases to cover the code you've written, where possible.
- Read the Contributing section of our documentation for more information around contributing to this project.

1.5 Running Tests

Does the code actually work?

```
$ pipenv install --dev
$ pipenv shell
(django_elastic_appsearch) $ tox
```

1.6 Credits

Tools used in rendering this package:

- Cookiecutter
- cookiecutter-djangopackage

CHAPTER 2

Installation

At the command line:

```
$ easy_install django_elastic_appsearch
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv django_elastic_appsearch
$ pip install django_elastic_appsearch
```

CHAPTER 3

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:

3.1 Types of Contributions

3.1.1 Report Bugs

Report bugs at https://github.com/CorrosiveKid/django_elastic_appsearch/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.

3.1.2 Fix Bugs

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

3.1.3 Implement Features

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

3.1.4 Write Documentation

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

3.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/CorrosiveKid/django_elastic_appsearch/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 :)

3.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/django_elastic_appsearch.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 django_elastic_appsearch
$ cd django_elastic_appsearch/
$ 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 django_elastic_appsearch 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.

3.3 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 3.4, 3.5, 3.6 and 3.7, and for Django > 2.0, and for PyPy. Check https://travis-ci.org/CorrosiveKid/django_elastic_appsearch/pull_requests and make sure that the tests pass for all supported Python versions.

3.4 Tips

To run a subset of tests:

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

CHAPTER 4

Credits

4.1 Development Lead

- Bianca Gibson <bianca.rachel.gibson@gmail.com>

4.2 Contributors

- Rasika Amaratissa <rasika.am@gmail.com>
- Mat Munn <mat@matmunn.me>

CHAPTER 5

History

5.1 1.1.5 (2021-05-11)

- Transfer ownership to Infoxchange
- Update documentation
- Dependency upgrades

5.2 1.1.4 (2021-05-05)

- Updated documentation
- Dependency upgrades

5.3 1.1.3 (2021-04-12)

- Add Django 3.2 to the test matrix
- Dependency upgrades
- Minor fixes

5.4 1.1.2 (2021-02-11)

- Security patch

5.5 1.1.1 (2021-02-04)

- Dependency upgrades
- Security patch
- Improve returned responses

5.6 1.1.0 (2021-01-27)

- Dependecy upgrades
- Add ability to index a model object into multiple app search engines

5.7 1.0.2 (2020-12-29)

- Dependency upgrades

5.8 1.0.1 (2020-12-15)

- Dependency upgrades

5.9 1.0.0 (2020-11-26)

- Python 3.9 support
- Update the project status to stable

5.10 0.7.6 (2020-11-26)

- Dependency upgrades
- Update documentation

5.11 0.7.5 (2020-10-28)

- Security patch

5.12 0.7.4 (2020-10-05)

- Add support for testing overridden queryset methods
- Update documentation

5.13 0.7.3 (2020-08-25)

- Remove support for Django 2.0 and Django 2.1
- Add support for Django 3.1
- Update documentation

5.14 0.7.2 (2020-08-25)

- Dependency upgrades

5.15 0.7.1 (2020-07-31)

- Dependency upgrades

5.16 0.7.0 (2020-07-30)

- Implement ability to do partial updates to documents
- Dependency upgrades

5.17 0.6.11 (2020-06-22)

- Fix failing dependency check with pipenv
- Dependency upgrades

5.18 0.6.9 (2020-05-15)

- Dependency upgrades

5.19 0.6.8 (2020-03-31)

- Dependency upgrades
- Security patches

5.20 0.6.7 (2020-02-25)

- Dependency upgrades

5.21 0.6.6 (2020-02-01)

- Dependency upgrades

5.22 0.6.3 (2020-01-03)

- Move from Travis CI to Github Actions
- Documentation updates

5.23 0.6.2 (2020-01-02)

- Dependency upgrades
- Documentation improvements
- Add linting for CI
- Setup automatic PyPI releases

5.24 0.6.1 (2019-12-24)

- Dependency upgrades

5.25 0.6.0 (2019-12-04)

- Remove support for Python 3.5
- Add support for Python 3.8
- Add support for Django 3
- Dependency upgrades
- Bump development status to Beta

5.26 0.5.6 (2019-12-03)

- Dependency upgrades

5.27 0.5.5 (2019-11-14)

- Dependency upgrades

5.28 0.5.4 (2019-10-02)

- Dependency upgrades

5.29 0.5.3 (2019-08-28)

- Improve documentation
- Refactor settings name `APPSEARCH_URL` -> `APPSEARCH_HOST`

5.30 0.5.1 (2019-08-26)

- Improve test coverage
- Improve documentation
- Add serpy as an official dependency
- Bump dependency versions
- Add code of conduct

5.31 0.4.2 (2019-08-16)

- Switch to the new official Elastic App Search python client
- Documentation improvements

5.32 0.2.3 (2019-08-02)

- Use Pipenv for dependency management
- Configure Dependabot for automatic dependency upgrades
- Remove support for Python 3.4
- Documentation improvements

5.33 0.2.2 (2019-07-29)

- Bug fixes
- Documentation improvements

5.34 0.1.0 (2019-07-26)

- First release on PyPI.