invenio-search Documentation

Release 1.4.0

CERN

Sep 18, 2020

Contents

Elasticsearch management for Invenio.

Features:

- Allows Invenio modules to register indexes, aliases and index templates.
- Manages the creation and deletion of indices, aliases and templates.
- API for providing stable searches (e.g. prevents bouncing of search results).
- Maps JSONSchema URLs to Elasticsearch indexes.
- Supports Elasticsearch v6 and v7.

Further documentation is available at https://invenio-search.readthedocs.io/.

CHAPTER 1

User's Guide

This part of the documentation will show you how to get started in using Invenio-Search.

1.1 Installation

Invenio-Search is on PyPI. When you install Invenio-Search you must specify the appropriate extras dependency for the version of Elasticsearch you use:

```
$ # For Elasticsearch 6.x:
$ pip install invenio-search[elasticsearch6]

$ # For Elasticsearch 7.x:
$ pip install invenio-search[elasticsearch7]
```

Elasticsearch v2 and v5 support still exists in Invenio-Search but will be deprecated in future releases so using v6 or v7 is recommended.

1.2 Configuration

The Elasticsearch client in Invenio is configured using the two configuration variables *SEARCH_CLIENT_CONFIG* and *SEARCH_ELASTIC_HOSTS*.

Invenio-Search relies on the following two Python packages to integrate with Elasticsearch:

- elasitcsearch
- elasitcsearch-dsl

1.2.1 Hosts

The hosts which the Elasticsearch client in Invenio should use are configured using the configuration variable:

invenio_search.config.**SEARCH_ELASTIC_HOSTS = None**
Elasticsearch hosts.

By default, Invenio connects to `localhost:9200`.

The value of this variable is a list of dictionaries, where each dictionary represents a host. The available keys in each dictionary is determined by the connection class:

- `elasticsearch.connection.Urllib3HttpConnection` (default)
- `elasticsearch.connection.RequestsHttpConnection`

You can change the connection class via the *SEARCH_CLIENT_CONFIG*. If you specified the `hosts` key in *SEARCH_CLIENT_CONFIG* then this configuration variable will have no effect.

Clusters

Normally in a production environment, you will run an Elasticsearch cluster on one or more dedicated nodes. Following is an example of how you configure Invenio to use such a cluster:

```
SEARCH_ELASTIC_HOSTS = [
    dict(host='es1.example.org'),
    dict(host='es2.example.org'),
    dict(host='es3.example.org'),
]
```

Elasticsearch will manage a connection pool to all of these hosts, and will automatically take nodes out if they fail.

Basic authentication and SSL

By default all traffic to Elasticsearch is via unencrypted HTTP because Elasticsearch does not come with built-in support for SSL unless you pay for the enterprise X-Pack addition. A cheaper alternative to X-Pack is to simply setup a proxy (e.g. nginx) on each node with SSL and HTTP basic authentication support.

Following is an example of how you configure Invenio to use SSL and Basic authentication when connecting to Elasticsearch:

```
params = dict(
    port=443,
    http_auth=('myuser', 'mypassword'),
    use_ssl=True,
)
SEARCH_ELASTIC_HOSTS = [
    dict(host='node1', **params),
    dict(host='node2', **params),
    dict(host='node3', **params),
]
```

Self-signed certificates

In case you are using self-signed SSL certificates on proxies in front of Elasticsearch, you will need to provide the `ca_certs` option:

```
params = dict(
    port=443,
    http_auth=('myuser', 'mypassword'),
    use_ssl=True,
    ca_certs='/etc/pki/tls/mycert.pem',
)
SEARCH_ELASTIC_HOSTS = [
    dict(host='node1', **params),
    # ...
]
```

Disabling SSL certificate verification

> **Warning:** We **strongly discourage** you to use this method. Instead, use the method with the `ca_certs` option documented above.
>
> Disabling verification of SSL certificates will e.g. allow man-in-the-middle attacks and give you a false sense of security (thus you could simply use plain unencrypted HTTP instead).

If you are using a self-signed certificate, you may also disable verification of the SSL certificate, using the `verify_certs` option:

```
import urllib3
urllib3.disable_warnings(
    urllib3.exceptions.InsecureRequestWarning
)

params = dict(
    port=443,
    http_auth=('myuser', 'mypassword'),
    use_ssl=True,
    verify_certs=False,
    ssl_show_warn=False, # only from 7.x+
)
SEARCH_ELASTIC_HOSTS = [
    dict(host='node1', **params),
    # ...
]
```

The above example will also disable the two warnings (`InsecureRequestWarning` and a `UserWarning`) using the `ssl_show_warn` option and urllib3 feature. Again, we **strongly discourage** you from using this method. The warnings are there for a reason!

Other host options

For a full list of options for configuring the hosts, see the connection classes documentation:

- `elasticsearch.connection.Urllib3HttpConnection` (default)
- `elasticsearch.connection.RequestsHttpConnection`

Other options include e.g.:

- `url_prefix`
- `client_cert`

- client_key

1.2.2 Client options

More advanced options for the Elasticsearch client are configured via the configuration variable:

invenio_search.config.**SEARCH_CLIENT_CONFIG = None**
Dictionary of options for the Elasticsearch client.

The value of this variable is passed to elasticsearch.Elasticsearch as keyword arguments and is used to configure the client. See the available keyword arguments in the two following classes:

- elasticsearch.Elasticsearch
- elasticsearch.Transport

If you specify the key hosts in this dictionary, the configuration variable *SEARCH_ELASTIC_HOSTS* will have no effect.

Timeouts

If you are running Elasticsearch on a smaller/slower machine (e.g. for development or CI) you might want to be a bit more relaxed in terms of timeouts and failure retries:

```
SEARCH_CLIENT_CONFIG = dict(
    timeout=30,
    max_retries=5,
)
```

Connection class

You can change the default connection class by setting the connection_class key (e.g. use requests library instead of urllib3):

```
from elasticsearch.connection import RequestsHttpConnection

SEARCH_CLIENT_CONFIG = dict(
    connection_class=RequestsHttpConnection
)
```

Note, that the default urllib3 connection class is more lightweight and performant than the requests library. Only use requests library for advanced features like e.g. custom authentication plugins.

Connection pooling

By default urllib3 will open up to 10 connections to each node. If your application calls for more parallelism, use the maxsize parameter to raise the limit:

```
SEARCH_CLIENT_CONFIG = dict(
    # allow up to 25 connections to each node
    maxsize=25,
)
```

Hosts via client config

Note, you may also use *SEARCH_CLIENT_CONFIG* instead of *SEARCH_ELASTIC_HOSTS* to configure the Elasticsearch hosts:

```
SEARCH_CLIENT_CONFIG = dict(
    hosts=[
        dict(host='es1.example.org'),
        dict(host='es2.example.org'),
        dict(host='es3.example.org'),
    ]
)
```

Other client options

For a full list of options for configuring the client, see the transport class documentation:

- elasticsearch.Elasticsearch
- elasticsearch.Transport

Other options include e.g.:

- url_prefix
- client_cert
- client_key

1.2.3 Index prefixing

Elasticsearch does not provide the concept of virtual hosts, and thus the only way to use a single Elasticsearch cluster with multiple Invenio instances is via prefixing index, alias and template names. This is defined via the configuration variable:

Warning: Note that index prefixing is only prefixing. Multiple Invenio instances sharing the same Elasticsearch cluster all have access to each other's indexes unless you use something like https://readonlyrest.com or the commercial X-Pack from Elasticsearch.

invenio_search.config.**SEARCH_INDEX_PREFIX = ''**

Any index, alias and templates will be prefixed with this string.

Useful to host multiple instances of the app on the same Elasticsearch cluster, for example on one app you can set it to *dev-* and on the other to *prod-*, and each will create non-colliding indices prefixed with the corresponding string.

Usage example:

```
# in your config.py
SEARCH_INDEX_PREFIX = 'prod-'
```

For templates, ensure that the prefix *__SEARCH_INDEX_PREFIX__* is added to your index names. This pattern will be replaced by the prefix config value.

Usage example in your template.json:

```
{
    "index_patterns": ["__SEARCH_INDEX_PREFIX__myindex-name-*"]
}
```

1.2.4 Index creation

Invenio will by default create all aliases and indexes registered into the `invenio_search.mappings` entry point. If this is not desirable for some reason, you can control which indexes are being created via the configuration variable:

`invenio_search.config.`**`SEARCH_MAPPINGS = None`**

List of aliases for which, their search mappings should be created.

- If *None* all aliases (and their search mappings) defined through the `invenio_search.mappings` entry point in setup.py will be created.
- Provide an empty list `[]` if no aliases (or their search mappings) should be created.

For example if you don't want to create aliases and their mappings for *authors*:

```
# in your `setup.py` you would specify:
entry_points={
    'invenio_search.mappings': [
        'records = invenio_foo_bar.mappings',
        'authors = invenio_foo_bar.mappings',
    ],
}

# and in your config.py
SEARCH_MAPPINGS = ['records']
```

1.3 Usage

Elasticsearch management for Invenio.

Allows retrieving records from a configurable backend (currently Elasticsearch is supported).

1.3.1 Initialization

To be able to retrieve information from *somewhere*, we first need to setup this *somewhere*. So make sure you have the correct version of Elasticsearch installed and running (see *Installation* for supported Elasticsearch versions).

For running an Elasticsearch instance we recommend using Docker and the official images provided by Elastic:

```
$ docker run -d \
    -p 9200:9200 \
    -e "discovery.type=single-node" \
    docker.elastic.co/elasticsearch/elasticsearch-oss:7.2.0
```

In this case, we are using Elasticsearch v7, so make sure to install `invenio-search` with the appropriate extras:

```
pip install invenio-search[elasticsearch7]
```

Creating index

To be able to run flask CLI commands later, we first need to have a Flask app, so create the following `app.py` file:

```
# app.py
from flask import Flask
from invenio_search import InvenioSearch

app = Flask('myapp')
search = InvenioSearch(app)
```

This will create an empty index, which is not very useful, so let's add some mappings. To not reinvent the wheel, let's reuse the mappings from the example application. Copy the `examples` directory from Invenio-Search. Add the following line at the end of `app.py` file that you just created:

```
# app.py
...
search.register_mappings('demo', 'examples.data')
```

The above code will search the directory `examples/data` and load all the mapping files it can find there for Elasticsearch. You can read more about the Elasticsearch mappings in the official documentation.

Now we can finally create the indexes. Each file in the mappings will create a new index and a top-level alias with the name `demo`.

```
$ export FLASK_APP=app.py
$ flask index init
```

You can verify that the indices were created correctly by performing the following requests:

```
$ # Fetch information about the "demo" alias
$ curl http://localhost:9200/demo
$ # Fetch information about the "demo-default-v1.0.0" alias
$ curl http://localhost:9200/demo-default-v1.0.0
```

Note: In earlier versions of Invenio-Search `demo-default-v1.0.0` was an index but is now a write alias pointing to a suffixed index. Read more about write aliases and suffixes in the *Aliases* section.

Let's index some data. Open `flask shell` and index a document with the following code:

```
import json
from invenio_search import current_search_client

current_search_client.index(
    index='demo-default-v1.0.0',
    body=json.dumps({
        'title': 'Hello invenio-search',
        'body': 'test 1'
    })
)
```

No error message? Good! You can see that your new document was indexed by going to http://localhost:9200/demo/_search.

Searching for data

Let's try to retrieve data in a programmatic way. Start a Python REPL and run the following commands.

First, let's initialize app from the `app.py` file that we created in the previous step. Python REPL.

```
from app import app
```

We will need a Flask application context, so let's push one:

```
app.app_context().push()
```

Create a custom search class that will search for `example` type of documents inside the `demo` alias (or you could use the default `RecordsSearch` class to search for all document types in all indexes):

```
from invenio_search import RecordsSearch
class ExampleSearch(RecordsSearch):
    class Meta:
        index = 'demo'
        fields = ('*', )
        facets = {}
search = ExampleSearch()
```

Let's find all documents:

```
response = search.execute()
response.to_dict()
```

If everything went well, you should now see that we have 1 hit - a document with `Hello invenio-search` title. If you get the `TransportError(404, 'index_not_found_exception', 'no such index')` error - it means that you forgot to create the index (follow the steps from *Creating index* to see how to setup an index and add example data).

Creating a search page

Let's create a simple web page where you can send queries to the Elasticsearch and see the results. Create a new app.py file with a route.

```
# app.py
from elasticsearch_dsl.query import QueryString
from flask import Flask, jsonify, request
from invenio_search import InvenioSearch, RecordsSearch

app = Flask('myapp')

search = InvenioSearch(app)

# This line is needed to be able to call `flask index init`
search.register_mappings('demo', 'examples.data')


@app.route('/', methods=['GET', 'POST'])
def index():
    search = RecordsSearch()
    if 'q' in request.values:
        search = search.query(QueryString(query=request.values.get('q')))
```

(continues on next page)

(continued from previous page)

```
    return jsonify(search.execute().to_dict())
```

Run example development server:

```
$ FLASK_DEBUG=1 FLASK_APP=app.py flask run -p 5000
```

And now you can perform search queries:

```
$ curl http://localhost:5000/?q=body:test
```

Filtering

To filter out some documents, you can create your own search class. Let's try to remove all private documents from the search results (by `private` documents, we understand all the documents that have `public` attribute set to 0).

Open `flask shell` and add one public and one private document to Elasticsearch:

```
import json
from invenio_search import current_search_client

# Index public document
current_search_client.index(
    index='demo-default-v1.0.0',
    body=json.dumps({
        'title': 'Public',
        'body': 'test 1',
        'public': 1
    })
)
# Index private document
current_search_client.index(
    index='demo-default-v1.0.0',
    body=json.dumps({
        'title': 'Private',
        'body': 'test 1',
        'public': 0
    })
)
```

Now, create a new search class that will return all documents of type `example` from the `demo` index and select only the public ones (documents where `public` is set to 1):

```
# app.py
from elasticsearch_dsl.query import Bool, Q, QueryString

class PublicSearch(RecordsSearch):
    class Meta:
        index = 'demo'
        fields = ('*', )
        facets = {}

    def __init__(self, **kwargs):
        super(PublicSearch, self).__init__(**kwargs)
        self.query = Q(
```

(continues on next page)

(continued from previous page)

```
            Bool(filter=[Q('term', public=1)])
        )
```

Update the `index` function and replace the search class with our new `PublicSearch` class:

```
# app.py
@app.route('/', methods=['GET', 'POST'])
def index():
    search = PublicSearch()
    ...
```

Now, you can search for documents with `test` in the body.

```
$ curl http://localhost:5000/?q=body:test
```

You should find only one document - the one with `Public` title.

This is a very simple example of how to filter out some records. If you want to define role based access rights control, check the invenio-access module.

1.3.2 Miscellaneous

Elasticsearch version support

Major versions of Elasticsearch can include breaking changes to mappings so mappings for each version of Elasticsearch are stored in separate folders. Invenio-Search will use these mappings when creating the indices. For backwards compatibility with existing Invenio modules and installations, Elasticsearch 2 mappings will be loaded from the root level of the package directory. You can see a full example in the `examples/data` directory of the Invenio-Search repository:

```
$ tree --dirsfirst examples/data

examples/data
+- demo            # Elasticsearch 2 mappings
|  +- authorities
|  |  +- authority-v1.0.0.json
|  +- bibliographic
|  |  +- bibliographic-v1.0.0.json
|  +- default-v1.0.0.json
+- v6
|  +- demo        # Elasticsearch 6 mappings
|  |  +- authorities
|  |  |  +- authority-v1.0.0.json
|  |  +- bibliographic
|  |  |  +- bibliographic-v1.0.0.json
|  |  +- default-v1.0.0.json
|  +- __init__.py
+- v7
|  +- demo        # Elasticsearch 7 mappings
|  |  +- authorities
|  |  |  +- authority-v1.0.0.json
|  |  +- bibliographic
|  |  |  +- bibliographic-v1.0.0.json
|  |  +- default-v1.0.0.json
```

(continues on next page)

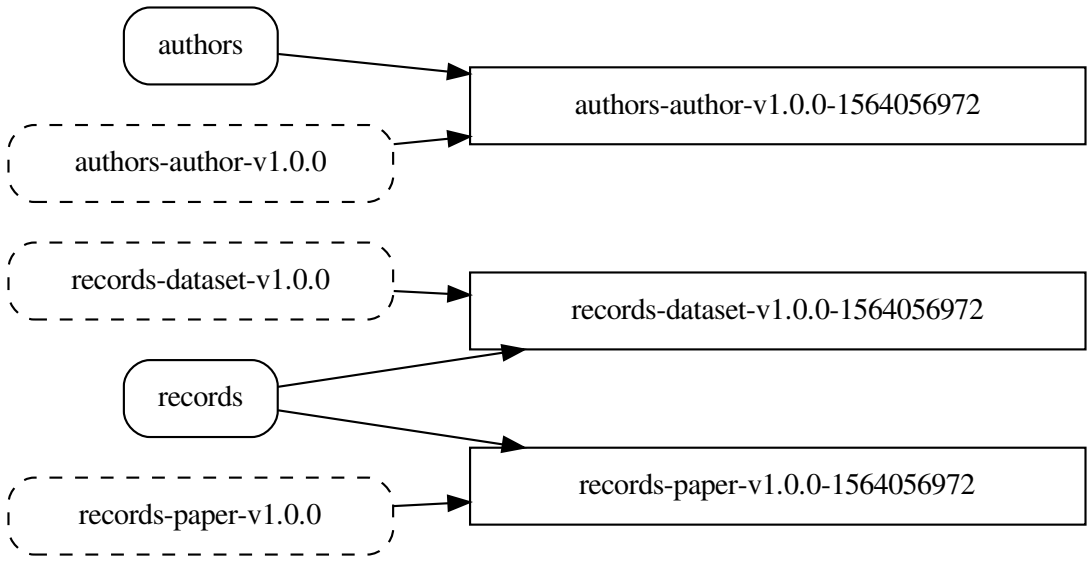
(continued from previous page)

```
|  +- __init__.py
+-- __init__.py
```

Elasticsearch plugins

For convenience, you can install a plugin like Elastic HQ for easy introspection of your indexes and their content. Otherwise, you can use curl as described in the official documentation.

Indexes and aliases



Indexes and aliases are organized as seen in the graph above. This example has three "concrete" indexes:

- `authors-author-v1.0.0-1564056972`
- `records-dataset-v1.0.0-1564056972`
- `records-paper-v1.0.0-1564056972`

They all share the suffix `1564056972`. Each index though has also a corresponding "write alias" with its un-suffixed name:

- `authors-author-v1.0.0 -> authors-author-v1.0.0-1564056972`
- `records-dataset-v1.0.0 -> records-dataset-v1.0.0-1564056972`
- `records-paper-v1.0.0 -> records-paper-v1.0.0-1564056972`

The other aliases in the example, `records` and `authors`, are top-level aliases pointing to all the indexes in their same hierarchy:

- `authors -> authors-author-v1.0.0-1564056972`

- `records -> records-dataset-v1.0.0-1564056972`
- `records -> records-paper-v1.0.0-1564056972`

Top-level **aliases** are aliases that can point to one or multiple indexes. The purpose of these aliases is to group indexes and be able to perform searches over multiple indexes. These aliases should never be indexed to as the indexing will fail if they point to multiple indexes.

The other type of alias is the **write alias** which is an alias that only points to a single index and has the same name as the index without the suffix. This alias should be used whenever you need to index something. The name of the write alias is the same as the un-suffixed index name to allow backwards compatibilty with previous versions of Invenio-Search.

An **index** ends with a suffix which is the timestamp of the index creation time. The suffix allows multiple revisions of the same index to exist at the same time. This is useful if you want to update the mappings of an index and migrate to a new index. With suffixes, it's possible to keep the two versions of the same index and sync them. When the migration is completed the write alias can be pointed to the new index and the application will use the new index. This allows in-cluster migrations without any downtime.

More information about index migrations can be found in the Invenio-Index-Migrator.

CHAPTER 2

API Reference

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

2.1 API Docs

Search engine API.

class invenio_search.api.**BaseRecordsSearch**(***kwargs*)

Example subclass for searching records using Elastic DSL.

Use Meta to set kwargs defaults.

class Meta

Configuration for `Search` and `FacetedSearch` classes.

default_filter = None

Default filter added to search body.

Example: `default_filter = DefaultFilter('_access.owner:"1"')`.

classmethod faceted_search(*query=None*, *filters=None*, *search=None*)

Return faceted search instance with defaults set.

Parameters

- **query** – Elastic DSL query object (`Q`).
- **filters** – Dictionary with selected facet values.
- **search** – An instance of `Search` class. (default: `cls()`).

get_record(*id_*)

Return a record by its identifier.

Parameters **id** – The record identifier.

Returns The record.

get_records(*ids*)
Return records by their identifiers.

Parameters **ids** – A list of record identifier.

Returns A list of records.

with_preference_param()
Add the preference param to the ES request and return a new Search.

The preference param avoids the bouncing effect with multiple replicas, documented on ES documentation. See: https://www.elastic.co/guide/en/elasticsearch/guide/current /_search_options.html#_preference for more information.

class invenio_search.api.**BaseRecordsSearchV2**(*fields=('*',), default_filter=None, **kwargs*)
Base records search V2.

Sets the needed args in kwargs for the search.

get_record(*id_*)
Return a record by its identifier.

Parameters **id** – The record identifier.

Returns The record.

get_records(*ids*)
Return records by their identifiers.

Parameters **ids** – A list of record identifier.

Returns A list of records.

with_preference_param(*preference*)
Add the preference param to the ES request and return a new Search.

The preference param avoids the bouncing effect with multiple replicas, documented on ES documentation. See: https://www.elastic.co/guide/en/elasticsearch/guide/current /_search_options.html#_preference for more information.

Parameters **preference** – A function that returns the preference value.

class invenio_search.api.**DefaultFilter**(*query=None*, *query_parser=None*)
Shortcut for defining default filters with query parser.

Build filter property with query parser.

query
Build lazy query if needed.

class invenio_search.api.**MinShouldMatch**
Work-around for Elasticsearch DSL problem.

The ElasticSearch DSL Bool query tries to inspect the `minimum_should_match` parameter, but understands only integers and not queries like "0<1". This class circumvents the specific problematic clause in Elasticsearch DSL.

class invenio_search.api.**PrefixedIndexList**
Custom list type for avoiding double prefixing.

class invenio_search.api.**PrefixedSearchMixin**
Mixing to use index prefixing.

prefix_index(*index*)
Using PrefixedIndexList type to avoid double prefixing.

class invenio_search.api.**RecordsSearch**(***kwargs*)
Prefixed record search class.

Constructor.

class invenio_search.api.**RecordsSearchV2**(***kwargs*)
Prefixed record search class.

Constructor.

invenio_search.api.**UnPrefixedRecordsSearch**
alias of *invenio_search.api.BaseRecordsSearch*

invenio_search.api.**UnPrefixedRecordsSearchV2**
alias of *invenio_search.api.BaseRecordsSearchV2*

2.1.1 Utilities

Utility functions for search engine.

invenio_search.utils.**build_alias_name**(*index*, *prefix=None*, *app=None*)
Build an alias name.

Parameters

- **index** – Name of the index.
- **prefix** – The prefix to prepend to the index name.

invenio_search.utils.**build_index_from_parts**(**parts*)
Build an index name from parts.

Parameters **parts** – String values that will be joined by dashes ("-").

invenio_search.utils.**build_index_name**(*index*, *prefix=None*, *suffix=None*, *app=None*)
Build an index name.

Parameters

- **index** – Name of the index.
- **prefix** – The prefix to prepend to the index name.
- **suffix** – The suffix to append to the index name.
- **app** – Flask app passed to `prefix_index` and `suffix_index`.

invenio_search.utils.**prefix_index**(*index*, *prefix=None*, *app=None*)
Prefixes the given index if needed.

Parameters

- **index** – Name of the index to prefix.
- **prefix** – Force a prefix.
- **app** – Flask app to get the prefix config from.

Returns A string with the new index name prefixed if needed.

invenio_search.utils.**schema_to_index**(*schema*, *index_names=None*)
Get index/doc_type given a schema URL.

Parameters

- **schema** – The schema name

- **index_names** – A list of index name.

Returns A tuple containing (index, doc_type).

invenio_search.utils.**suffix_index**(*index*, *suffix=None*, *app=None*)
Suffixes the given index.

Parameters

- **index** – Name of the index to prefix.
- **suffix** – The suffix to append to the index name.
- **app** – Flask app to get the "invenio-search" extension from.

Returns A string with the new index name suffixed.

invenio_search.utils.**timestamp_suffix**()
Generate a suffix based on the current time.

CHAPTER 3

Additional Notes

Notes on how to contribute, legal information and changes are here for the interested.

3.1 Contributing

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

3.1.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/inveniosoftware/invenio-search/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.

Fix Bugs

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

Implement Features

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

Write Documentation

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

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/inveniosoftware/invenio-search/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 :)

3.1.2 Get Started!

Ready to contribute? Here's how to set up *invenio-search* for local development.

1. Fork the *inveniosoftware/invenio-search* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/invenio-search.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 invenio-search
$ cd invenio-search/
$ pip install -e .[all]
```

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 tests:

```
$ ./run-tests.sh
```

   The tests will provide you with test coverage and also check PEP8 (code style), PEP257 (documentation), flake8 as well as build the Sphinx documentation and run doctests.

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

```
$ git add .
$ git commit -s
    -m "component: title without verbs"
    -m "* NEW Adds your new feature."
    -m "* FIX Fixes an existing issue."
    -m "* BETTER Improves and existing feature."
    -m "* Changes something that should not be visible in release notes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

3.1.3 Pull Request Guidelines

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

1. The pull request should include tests and must not decrease test coverage.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring.
3. The pull request should work for Python 2.7, 3.3, 3.4 and 3.5. Check https://travis-ci.org/inveniosoftware/invenio-search/pull_requests and make sure that the tests pass for all supported Python versions.

3.2 Changes

Version 1.4.0 (released 2020-09-18)

- Adds new search class that can be initialised only from arguments to the constructor.

Version 1.3.1 (released 2020-05-07)

- Set Sphinx `<3.0.0` because of errors related to application context.
- Stop using example app, keep only files referenced in the docs.

Version 1.3.0 (released 2020-03-10)

- Centralize dependency management via Invenio-Base.

Version 1.2.4 (released 2020-05-07)

- Set Sphinx `<3.0.0` because of errors related to application context.
- Stop using example app, keep only files referenced in the docs.

Version 1.2.3 (released 2019-10-07)

- Changes the naming strategy for templates to avoid inclusion of slashes ("/")

Version 1.2.2 (released 2019-08-08)

- Adds option ignore_existing which is ignoring indexes which are already in ES.
- Adds option to create/delete only selected indexes.

Version 1.2.1 (released 2019-07-31)

- Unpins `urllib3` and `idna` since `requests` is not a direct dependency of the package now.

Version 1.2.0 (released 2019-07-29)

- Adds full Elasticsearch v7 support
- Better prefixing integration
- Introduces index suffixes and write aliases
- Refactored the way indices and aliases are stored and created
- `invenio_search.utils.schema_to_index` is deprecated (moved to `invenio-indexer`)
- Deprecates Elasticsearch v5

Version 1.1.1 (released 2019-06-25)

- Fixes prefixing for whitelisted aliases and the RecordSearch class.
- Adds basic Elasticsearch v7 support.

Version 1.1.0 (released 2019-02-25)

- Deprecates Elasticsearch v2
- Adds support for Elasticsearch indices prefix

Version 1.0.2 (released 2018-10-23)

- Updates the urllib3 dependency version pin.
- Pins elasticsearch-dsl to <6.2.0, because of a breaking change in the handling of empty queries.
- Adds the SEARCH_CLIENT_CONFIG configuration variable, allowing more complex configuration to be passed to the Elasticsearch client initialization.

Version 1.0.1 (released 2018-06-13)

- Fixes issues with idna/urllib3 dependencies conflicts.
- Adds SEARCH_RESULTS_MIN_SCORE configuration variable to allow excluding search results which have a score less than the specified value.

Version 1.0.0 (released 2018-03-23)

- Initial public release.

3.3 License

Note: In applying this license, CERN does not waive the privileges and immunities granted to it by virtue of its status as an Intergovernmental Organization or submit itself to any jurisdiction.

3.4 Contributors

- Alexander Ioannidis
- Alizee Pace
- Chiara Bigarella

- Chris Aslanoglou
- David Caro
- Dinos Kousidis
- Esteban J. G. Gabancho
- Harri Hirvonsalo
- Harris Tzovanakis
- Javier Delgado
- Jiri Kuncar
- Lars Holm Nielsen
- Leonardo Rossi
- Nicola Tarocco
- Nicolas Harraudeau
- Nikos Filippakis
- Paulina Lach
- Sami Hiltunen
- Sebastian Witowski
- Tibor Simko

Python Module Index

i

Index

T

U

W