invenio-indexer Documentation

Release 1.1.2

CERN

Apr 28, 2020

Contents

Record indexer for Invenio.

Further documentation is available on https://invenio-indexer.readthedocs.io/

CHAPTER 1

User's Guide

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

1.1 Installation

Invenio-Indexer is on PyPI so all you need is:

```
$ pip install invenio-indexer
```

Invenio-Indexer depends on Invenio-Search, Invenio-Records and Celery/Kombu.

Requirements

Invenio-Indexer requires a message queue in addition to Elasticsearch (Invenio-Search) and a database (Invenio-Records). See Kombu documentation for list of supported message queues (e.g. RabbitMQ): http://kombu.readthedocs.io/en/latest/introduction.html#transport-comparison

1.2 Configuration

Record indexer for Invenio.

`invenio_indexer.config.`**`INDEXER_BEFORE_INDEX_HOOKS`** **= []**
List of automatically connected hooks (function or importable string).

`invenio_indexer.config.`**`INDEXER_BULK_REQUEST_TIMEOUT`** **= 10**
Request timeout to use in Bulk indexing.

`invenio_indexer.config.`**`INDEXER_DEFAULT_DOC_TYPE`** **= 'record-v1.0.0'**
Default doc_type to use if no schema is defined.

`invenio_indexer.config.`**`INDEXER_DEFAULT_INDEX`** **= 'records-record-v1.0.0'**
Default index to use if no schema is defined.

invenio_indexer.config.**INDEXER_MQ_EXCHANGE = <unbound Exchange indexer(direct)>**
Default exchange for message queue.

invenio_indexer.config.**INDEXER_MQ_QUEUE = <unbound Queue indexer -> <unbound Exchange indexer(direct)> ->**
Default queue for message queue.

invenio_indexer.config.**INDEXER_MQ_ROUTING_KEY = 'indexer'**
Default routing key for message queue.

invenio_indexer.config.**INDEXER_RECORD_TO_INDEX = 'invenio_indexer.utils.default_record_to_index'**
Provide an implemetation of record_to_index function

invenio_indexer.config.**INDEXER_REPLACE_REFS = True**
Whether to replace JSONRefs prior to indexing record.

1.3 Usage

Record indexer for Invenio.

Invenio-Indexer is responsible for sending records for indexing in Elasticsearch so that the records can be searched. Invenio-Indexer can either send the records in bulk or individually. Bulk indexing is far superior in performance if multiple records needs to be indexed at the price of delay. Bulk indexing works by queuing records in a message queue, which is then consumed and sent to Elasticsearch.

1.3.1 Initialization

First create a Flask application:

```
>>> from flask import Flask
>>> app = Flask('myapp')
>>> app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite://'
```

You initialize Indexer like a normal Flask extension, however Invenio-Indexer is dependent on Invenio-Records and Invenio-Search so you need to initialize these extensions first:

```
>>> from invenio_db import InvenioDB
>>> ext_db = InvenioDB(app)
>>> from invenio_search import InvenioSearch
>>> ext_search = InvenioSearch(app)
>>> from invenio_records import InvenioRecords
>>> ext_records = InvenioRecords(app)
```

We now initialize Invenio-Indexer:

```
>>> from invenio_indexer import InvenioIndexer
>>> ext_indexer = InvenioIndexer(app)
```

In order for the following examples to work, you need to work within an Flask application context so let's push one:

```
>>> ctx = app.app_context()
>>> ctx.push()
```

Also, for the examples to work we need to create the database tables and Elasticsearch indexes (note, in this example we use an in-memory SQLite database):

```
>>> from invenio_db import db
>>> db.create_all()
```

1.3.2 Indexing a record

Let's start by creating a record that we would like to index:

```
>>> from invenio_db import db
>>> from invenio_records.api import Record
>>> record = Record.create({'title': 'A test'})
>>> db.session.commit()
```

Note, that you are responsible for ensuring that the record is committed to the database, prior to sending it for indexing.

Now, let's index the record:

```
>>> from invenio_indexer.api import RecordIndexer
>>> indexer = RecordIndexer()
>>> res = indexer.index(record)
```

By default, records are sent to the Elasticsearch index defined by the configuration variable `INDEXER_DEFAULT_INDEX`. If the record however has a `$schema` attribute, the index is automatically determined from this. E.g. the following record:

```
>>> r = Record({
...     '$schema': 'http://example.org/records/record-v1.0.0.json'})
```

Would be indexed in the following Elasticsearch index/doctype:

```
>>> index, doc_type = indexer.record_to_index(record)
```

1.3.3 Bulk indexing

If you have many records to index, bulk indexing is far superior in speed to single record indexing. Bulk indexing requires the existence of a queue on your broker, so since this is the very first time we send any records for bulk indexing, we will have to create this queue:

```
>>> from celery.messaging import establish_connection
>>> queue = app.config['INDEXER_MQ_QUEUE']
>>> with establish_connection() as conn:
...     queue(conn).declare()
'indexer'
```

We can now send a record for bulk indexing:

```
>>> indexer.bulk_index([str(r.id)])
```

Above will send the record id to the queue on your broker and wait for the bulk indexer to execute. This is normally done in the background by a Celery task which can be started from the command line like e.g.:

```
$ <instance cmd> index run
```

Note, you can achieve much higher indexing speeds, by having multiple processes running `process_bulk_queue` concurrently. This can be achieved with e.g.:

```
# Send 8 Celery tasks to bulk index messages from the "indexer" queue
$ <instance cmd> index run -d -c 8
```

1.3.4 Customizing record indexing

Record indexing can easily be customized using either:

- **JSONRef:** By default, all JSONRefs for each record is resolved prior to indexing.
- **Signals:** Before each record is indexed the signal `before_record_index` is sent, in order to allow modification of the record. The signal can be used to e.g. remove sensitive data and/or add extra data to the record.

JSONRef

JSONRefs inside the record are by default resolved prior to indexing the record. For instance the value for the `rel` key will be replaced with the referenced JSON object:

```
>>> r = Record.create({
...     'title': 'A ref',
...     'rel': {'$ref': 'http://dx.doi.org/10.1234/foo'}})
```

See Invenio-Records documentation for how to customize the JSONRef resolver to resolve references locally. The JSONRefs resolving works on all indexed records, and can be switched off using the configuration:

```
>>> app.config['INDEXER_REPLACE_REFS'] = False
```

Signal

First write a signal receiver. In the example below, we remove the attribute `_internal` if it exists in the record:

```
>>> def indexer_receiver(sender, json=None, record=None,
...                      index=None, doc_type=None, arguments=None, **kwargs):
...     if '_internal' in json:
...         del json['_internal']
```

The receiver takes various parameters besides the sender (which is the Flask application)

- `json`: JSON is a Python dictionary dump of the record, and the actual data that will be sent to the index. Modify this dictionary in order to change the document.
- `record`: The record from which the JSON was dumped.
- `index`: The Elasticsearch index in which the record will be indexed.
- `doc_type`: The Elasticsearch document type for the record.
- `arguments`: The arguments that will be passed to the `index()` call.

Connecting the receiver to the signal is as simple as (do this e.g. in your extension's `init_app` method):

```
>>> from invenio_indexer.signals import before_record_index
>>> res = before_record_index.connect(indexer_receiver, sender=app)
```

Receivers can be useful if you have rules that apply to all of your records. If specific types of records have different rules (e.g. in case you had "records" and "authors") you can use the `before_record_index.dynamic_connect()` function as so:

```
>>> # Only be applied to documents sent to the "authors-v1.0.0" index
>>> res = before_record_index.dynamic_connect(
...     indexer_receiver, sender=app, index='authors-v1.0.0')
```

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

2.1.1 Record Indexer

API for indexing of records.

class `invenio_indexer.api.`**`BulkRecordIndexer`**(*search_client=None*, *exchange=None*, *queue=None*, *routing_key=None*, *version_type=None*, *record_to_index=None*)

Provide an interface for indexing records in Elasticsearch.

Uses bulk indexing by default.

Initialize indexer.

Parameters

- **`search_client`** – Elasticsearch client. (Default: `current_search_client`)
- **`exchange`** – A `kombu.Exchange` instance for message queue.
- **`queue`** – A `kombu.Queue` instance for message queue.
- **`routing_key`** – Routing key for message queue.
- **`version_type`** – Elasticsearch version type. (Default: `external_gte`)
- **`record_to_index`** – Function to extract the index and doc_type from the record.

`delete`(*record*)

Delete a record.

Parameters **`record`** – Record instance.

delete_by_id(*record_uuid*)
Delete record from index by record identifier.

index(*record*)
Index a record.

The caller is responsible for ensuring that the record has already been committed to the database. If a newer version of a record has already been indexed then the provided record will not be indexed. This behavior can be controlled by providing a different `version_type` when initializing `RecordIndexer`.

Parameters **record** – Record instance.

index_by_id(*record_uuid*)
Index a record by record identifier.

Parameters **record_uuid** – Record identifier.

class `invenio_indexer.api.`**Producer**(*channel*, *exchange=None*, *routing_key=None*, *serializer=None*, *auto_declare=None*, *compression=None*, *on_return=None*)
Producer validating published messages.

For more information visit `kombu.Producer`.

publish(*data*, ***kwargs*)
Validate operation type.

class `invenio_indexer.api.`**RecordIndexer**(*search_client=None*, *exchange=None*, *queue=None*, *routing_key=None*, *version_type=None*, *record_to_index=None*)
Provide an interface for indexing records in Elasticsearch.

Bulk indexing works by queuing requests for indexing records and processing these requests in bulk.

Initialize indexer.

Parameters

- **search_client** – Elasticsearch client. (Default: `current_search_client`)
- **exchange** – A `kombu.Exchange` instance for message queue.
- **queue** – A `kombu.Queue` instance for message queue.
- **routing_key** – Routing key for message queue.
- **version_type** – Elasticsearch version type. (Default: `external_gte`)
- **record_to_index** – Function to extract the index and doc_type from the record.

bulk_delete(*record_id_iterator*)
Bulk delete records from index.

Parameters **record_id_iterator** – Iterator yielding record UUIDs.

bulk_index(*record_id_iterator*)
Bulk index records.

Parameters **record_id_iterator** – Iterator yielding record UUIDs.

create_producer(**args*, ***kwds*)
Context manager that yields an instance of `Producer`.

delete(*record*, ***kwargs*)
Delete a record.

Parameters

- **record** – Record instance.
- **kwargs** – Passed to `elasticsearch.Elasticsearch.delete()`.

delete_by_id(*record_uuid*, ***kwargs*)

Delete record from index by record identifier.

Parameters

- **record_uuid** – Record identifier.
- **kwargs** – Passed to `RecordIndexer.delete()`.

index(*record*, *arguments=None*, ***kwargs*)

Index a record.

The caller is responsible for ensuring that the record has already been committed to the database. If a newer version of a record has already been indexed then the provided record will not be indexed. This behavior can be controlled by providing a different `version_type` when initializing `RecordIndexer`.

Parameters **record** – Record instance.

index_by_id(*record_uuid*, ***kwargs*)

Index a record by record identifier.

Parameters

- **record_uuid** – Record identifier.
- **kwargs** – Passed to `RecordIndexer.index()`.

mq_exchange

Message Queue exchange.

Returns The Message Queue exchange.

mq_queue

Message Queue queue.

Returns The Message Queue queue.

mq_routing_key

Message Queue routing key.

Returns The Message Queue routing key.

process_bulk_queue(*es_bulk_kwargs=None*)

Process bulk indexing queue.

Parameters **es_bulk_kwargs** (*dict*) – Passed to `elasticsearch.helpers.bulk()`.

record_cls

alias of `Record`

record_to_index(*record*)

Get index/doc_type given a record.

Parameters **record** – The record where to look for the information.

Returns A tuple (index, doc_type).

2.1.2 Flask Extension

Flask exension for Invenio-Indexer.

class `invenio_indexer.ext.`**`InvenioIndexer`**(*app=None*)

Invenio-Indexer extension.

Extension initialization.

Parameters **`app`** – The Flask application. (Default: `None`)

`init_app`(*app*)

Flask application initialization.

Parameters **`app`** – The Flask application.

`init_config`(*app*)

Initialize configuration.

Parameters **`app`** – The Flask application.

`record_to_index`

Import the configurable 'record_to_index' function.

2.1.3 Celery tasks

Celery tasks to index records.

`invenio_indexer.tasks.`**`delete_record`**(*record_uuid*)

Delete a single record.

Parameters **`record_uuid`** – The record UUID.

`invenio_indexer.tasks.`**`index_record`**(*record_uuid*)

Index a single record.

Parameters **`record_uuid`** – The record UUID.

`invenio_indexer.tasks.`**`process_bulk_queue`**(*version_type=None*, *es_bulk_kwargs=None*)

Process bulk indexing queue.

Parameters

- **`version_type`** (*`str`*) – Elasticsearch version type.
- **`es_bulk_kwargs`** (*`dict`*) – Passed to `elasticsearch.helpers.bulk()`.

Note: You can start multiple versions of this task.

`invenio_indexer.tasks.`**`process_bulk_queue`**(*version_type*)

Process bulk indexing queue.

Parameters

- **`version_type`** (*`str`*) – Elasticsearch version type.
- **`es_bulk_kwargs`** (*`dict`*) – Passed to `elasticsearch.helpers.bulk()`.

Note: You can start multiple versions of this task.

`invenio_indexer.tasks.`**`index_record`**(*record_uuid*)

Index a single record.

Parameters **`record_uuid`** – The record UUID.

`invenio_indexer.tasks.`**`delete_record`**(*record_uuid*)

Delete a single record.

Parameters **`record_uuid`** – The record UUID.

2.1.4 Signals

Signals for indexer.

`invenio_indexer.signals.`**`before_record_index`** **= <blinker.base.NamedSignal object at 0x7fc8cd6328d0; 'before-re**

Signal sent before a record is indexed.

The sender is the current Flask application, and two keyword arguments are provided:

- `json`: The dumped record dictionary which can be modified.
- `record`: The record being indexed.
- `index`: The index in which the record will be indexed.
- `doc_type`: The doc_type for the record.
- `arguments`: The arguments to pass to Elasticsearch for indexing.
- `**kwargs`: Extra arguments.

This signal also has a `.dynamic_connect()` method which allows some more flexible ways to connect receivers to it. The most common use case is that you want to apply a receiver only to a specific index. In that case you can call:

For more complex conditions you can provide a function via the `condition_func` parameter like so:

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-indexer/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-Indexer could always use more documentation, whether as part of the official Invenio-Indexer 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-indexer/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-indexer* for local development.

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

```
$ git clone git@github.com:your_name_here/invenio-indexer.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-indexer
$ cd invenio-indexer/
$ 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, 3.5 and 3.6. Check https://travis-ci.org/inveniosoftware/invenio-indexer/pull_requests and make sure that the tests pass for all supported Python versions.

3.2 Changes

Version 1.1.2 (released 2020-04-28)

- Introduces `RecordIndexer.record_cls` for customizing the record class.
- Removes Python 2 support.

Version 1.1.1 (released 2019-11-21)

- Fix bulk action parameters compatibility for Elasticsearch v7.

Version 1.1.0 (released 2019-07-19)

- Add support for Elasticsearch v7.
- Integrate index prefixing.
- Add `before_record_index.dynamic_connect()` signal utility for more flexible indexer receivers.
- Add `schema_to_index` utility from `invenio-search` (will be removed in next minor version of `invenio-search`).

Version 1.0.2 (released 2019-05-27)

- Allow Elasticsearch indexing arguments to be modified by subscribing to `before_record_index` signal.

Version 1.0.1 (released 2018-10-11)

- Allow forwarding arguments from `RecordIndexer.process_bulk_queue` to `elasticsearch.helpers.bulk` calls via the `es_bulk_kwargs` parameter.

Version 1.0.0 (released 2018-03-23)

- Initial public release.

3.3 License

3.4 Contributors

- Alexander Ioannidis
- Alizee Pace
- Chiara Bigarella
- Diego Rodriguez
- Harris Tzovanakis
- Javier Martin Montull
- Jiri Kuncar
- Krzysztof Nowak
- Lars Holm Nielsen
- Leonardo Rossi
- Nicolas Harraudeau
- Nikos Filippakis
- Paulina Lach
- Salvatore Zaza
- Sebastian Witowski
- Tibor Simko

Python Module Index

i

Index