invenio-cache Documentation

Release 1.1.0

CERN

Apr 29, 2020

Contents

Cache module for Invenio.

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

CHAPTER 1

User's Guide

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

1.1 Installation

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

```
$ pip install invenio-cache
```

Note, depending on which cache backend you plan yo use, you need to install extra modules. For instance for Redis you need:

```
$ pip install redis
```

For memcached you need either pylibmc or python-memcached installed:

```
$ pip install pylibmc
$ pip install python-memcached
```

1.2 Configuration

Configuration for Invenio-Cache module.

By default the module is configured to use a Redis database 0 on localhost. The underlying Flask-Caching module however supports many other cahe backends.

For how to configure other cache backends please refer to the Flask-Caching documentation.

`invenio_cache.config.`**`CACHE_IS_AUTHENTICATED_CALLBACK = None`**

Import path to callback.

Callback is executed to determine if request is authenticated.

invenio_cache.config.**CACHE_KEY_PREFIX = 'cache::'**
Cache key prefix.

invenio_cache.config.**CACHE_REDIS_URL = 'redis://localhost:6379/0'**
Redis location and database.

invenio_cache.config.**CACHE_TYPE = 'redis'**
Cache type.

Please refer to Flask-Caching documentation for other cache types.

1.3 Usage

Cache module for Invenio.

1.3.1 Initialization

Create a Flask application:

```
>>> from flask import Flask
>>> app = Flask('myapp')
>>> app.config['CACHE_TYPE'] = 'simple'
```

Initialize Invenio-Cache:

```
>>> from invenio_cache import InvenioCache
>>> ext = InvenioCache(app)
```

1.3.2 Jinja bytecode caching

By default Jinja only supports filesystem and memcached backends for bytecode caching. Invenio-Cache provides another backend, which will use the default configured cache backend instead (and thus e.g. supports Redis). Bytecode caching helps reduce template load time especially in a multi-process environment where workers are recycled from time to time.

For more information about Jinja bytecode caching please see http://jinja.pocoo.org/docs/2.9/api/#bytecode-cache

Enabling the bytecode cache is as simple as:

```
>>> from invenio_cache import BytecodeCache
>>> app.jinja_options = dict(
...     app.jinja_options,
...     cache_size=1000,
...     bytecode_cache=BytecodeCache(app)
... )
```

1.3.3 Programmatic API

The programmatic cache API is very simple. First get your cache instance:

```
>>> cache = ext.cache
```

If you are in an Flask application context you can also use a handy proxy:

```
>>> app.app_context().push()
>>> from invenio_cache import current_cache
```

Now, simply set, get and delete cache values:

```
>>> current_cache.set('mykey', 'myvalue')
True
>>> current_cache.get('mykey')
'myvalue'
>>> current_cache.delete('mykey')
True
```

1.3.4 Further documentation

Flask-Caching has a good and extensive API documentation so please refer to that for other APIs such as cache decorators for view, memoization of functions, Jinja snippet caching.

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

Cache module for Invenio.

class invenio_cache.ext.**InvenioCache**(*app=None*)
Invenio-Cache extension.

Extension initialization.

init_app(*app*)
Flask application initialization.

init_config(*app*)
Initialize configuration.

2.1.1 Decorators

Decorators to help with caching.

invenio_cache.decorators.**cached_unless_authenticated**(*timeout=50*, *key_prefix='default'*)
Cache anonymous traffic.

2.1.2 Proxies

Helper proxies.

invenio_cache.proxies.**current_cache = <LocalProxy unbound>**
Helper proxy to access cache object.

invenio_cache.proxies.**current_cache_ext = <LocalProxy unbound>**
Helper proxy to access cache extension object.

2.1.3 Bytecode cache

Jinja bytecode cache for Redis.

class invenio_cache.bccache.**BytecodeCache**(*app*)
A bytecode cache.

Initialize *BytecodeCache*.

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

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

```
$ git clone git@github.com:your_name_here/invenio-cache.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-cache
$ cd invenio-cache/
$ 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-cache/pull_requests and make sure that the tests pass for all supported Python versions.

3.2 Changes

Version 1.1.0 (released 2020-03-10)

- changes flask dependency to centrally managed

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

- Lars Holm Nielsen

Python Module Index

i

Index

B

C

I