Celery Persistent Revokes Documentation

Release 1.0.1

Hugo Bessa

Aug 03, 2018

Contents

Contents:

CHAPTER 1

Celery Persistent Revokes

Celery task revokes are stored on memory or on file. This packages makes possible to easily customize how your revokes are stored (Ex.: Database).

1.1 Documentation

The full documentation is at https://celery-persistent-revokes.readthedocs.io.

1.2 Quickstart

Install Celery Persistent Revokes:

```
pip install celery-persistent-revokes
```

Add it to your *INSTALLED_APPS*:

```
INSTALLED_APPS = (
    ...
    'celery_persistent_revokes.apps.CeleryPersistentRevokesConfig',
    ...
)
```

Add Celery Persistent Revokes's URL patterns:

```
from celery_persistent_revokes import urls as celery_persistent_revokes_urls


urlpatterns = [
    ...
    url(r'^', include(celery_persistent_revokes_urls)),
```

(continues on next page)

(continued from previous page)

```
    ...
]
```

1.3 Features

- Built-in tasks revokes using Django default database
- Support for custom backends

1.4 Running Tests

Does the code actually work?

```
source <YOURVIRTUALENV>/bin/activate
(myenv) $ pip install tox
(myenv) $ tox
```

1.5 Credits

Tools used in rendering this package:

- Cookiecutter
- cookiecutter-djangopackage

CHAPTER 2

Installation

At the command line:

```
$ easy_install celery-persistent-revokes
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv celery-persistent-revokes
$ pip install celery-persistent-revokes
```

CHAPTER 3

Usage

3.1 Django Projects

To use Celery Persistent Revokes in a Django project, add it to your *INSTALLED_APPS*:

```
INSTALLED_APPS = (
    ...
    'celery_persistent_revokes.apps.CeleryPersistentRevokesConfig',
    ...
)
```

Run migrations:

Use `revokable_task()` decorator to create tasks:

```
from celery_persistent_revokes.decorators import revokable_task


@revokable_task(my_celery_app)
def my_pretty_task(my_arg):
    do_something(my_arg)
    return
```

Use `revoke()` helper to create tasks:

```
from celery_persistent_revokes.helpers import revoke

result = my_pretty_task.delay()

# ...

revoke(result.id)
```

3.2 Other Python Projects

To use Celery Persistent Revokes without Django, you need to implement your own task management backend. But don't panic, this is pretty straight forward:

```
class MyCustomBackend(object):

    def revoke(self, task_id):
        # save on your storage the id of the task to be revoked

    def list_revokes(self):
        # list the ids of your revoked tasks

    def is_task_revoked(self, task_id):
        # returns True if the task with the id equal to task_id
        # is marked as revoked in your storage

    def delete_revoke(self, task_id):
        # removes the revoked task with id equal to task_id from
        # your storage
```

Then you just need to set the path to MyCustomBackend in an environment variable called CELERY_PERSISTENT_REVOKES_BACKEND.

Supposing that you have the following file structure and MyCustomBackend is in a task_revoke_backends module:

```
$ export CELERY_PERSISTENT_REVOKES_BACKEND='task_revoke_backends.MyCustomBackend'
```

CHAPTER 4

Configuration

Configurations can be set as environment variables and as Django settings too (in case your project uses Django.

The variables are:

CELERY_PERSISTENT_REVOKES_BACKEND: Default value: 'celery_persistent_revokes.backends.DjangoDatabase'. This variable defines the backend used to store and fetch the tasks ids of the tasks you revoke using this package.

CELERY_PERSISTENT_REVOKES_MODEL: Default value: 'celery_persistent_revokes.CeleryTaskRevoke'. If you're using DjangoDatabase backend, you can use this variable to define another Django model to store your Revokes.

```
from django.db import models
from celery_persistent_revokes.models import CeleryTaskRevoke

class MyCustomRevoke(CeleryTaskRevoke)
    created = models.DateTimeField(auto_now=True)
```

CHAPTER 5

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:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/vintasoftware/celery-persistent-revokes/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.

5.1.2 Fix Bugs

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

5.1.3 Implement Features

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

5.1.4 Write Documentation

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

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/vintasoftware/celery-persistent-revokes/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 :)

5.2 Get Started!

Ready to contribute? Here's how to set up *celery-persistent-revokes* for local development.

1. Fork the *celery-persistent-revokes* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/celery-persistent-revokes.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 celery-persistent-revokes
$ cd celery-persistent-revokes/
$ pip install -r requirements/dev.txt
$ 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 prospector and the tests, including testing other Python versions with tox:

```
$ prospector
$ python setup.py test
$ make test-all
```

   To get prospector 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
```

Before every commit, pre-commit runs some checks to make sure the changes are according to the project's code style.

If, for some reason, you need to commit without running the checks, you can skip them by using a -n flag:

```
$ git commit -m "Your detailed description of your changes." -n
```

7. Submit a pull request through the GitHub website.

5.3 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 2.7, 3.4, 3.5 and 3.6. Check https://travis-ci.org/vintasoftware/celery-persistent-revokes/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ python runtests.py tests.test_celery_persistent_revokes
```

CHAPTER 6

Credits

6.1 Development Lead

- Hugo Bessa <hugo@vinta.com.br>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

Change Log

1.0.1 (2018-04-03)

- Adds warning in logs when a revoked tasks is run by a worker

1.0.0 (2018-04-03)

- Updates Django revokes model to be timestamped

7.1 0.1.0 (2018-04-25)

- First release on PyPI.