Transactional Celery for Pyramid Documentation

Release 0.1.0

Christopher Petrilli

January 20, 2015

Contents

Contents:

CHAPTER 1

Transactional Celery for Pyramid

A transaction-aware Celery job setup. This is integrated with the Zope transaction package, which implements a full two-phase commit protocol. While it is not designed for anything other than Pyramid, it also does not use any component of Pyramid. It's simply not tested anywhere else.

- Free software: BSD license
- Documentation: https://pyramid_transactional_celery.readthedocs.org.

1.1 Features

- Queues tasks into a thread-local when they are called either using `delay` or `apply_async`.
- If the transaction is aborted, then the tasks will never be called.
- If the transaction is committed, the tasks will go through their normal `apply_async` process and be queued for processing.

1.2 Limitations

Currently, the code is designed around Celery v3.1, and it is unknown whether it will work with previous versions. I'm more than happy to integrate changes that would make it work with other releases, but since I generally stay on the latest release, it isn't a priority for my own development.

1.3 Usage

Using the library is a relatively easy thing to do. First, you'll need to integrate Celery into your Pyramid application, for which I recommend using pyramid_celery. Once that's done, you simply need to start creating your tasks. The big difference is for function-based tasks, you use a different decorator:

```
from pyramid_transactional_celery import task_tm

@task_tm
def add(x, y):
    """Add two numbers together."""
    return x + y
```

That's all there is to it. For class-based tasks, you simply need to subclass `TransactionalTask` instead of `Task`:

```
from pyramid_transactional_celery import TransactionalTask

class SampleTask(TransactionalTask):
    """A sample task that is transactional."""
    def run(x, y):
        return x + y
```

That's it. Bob's your uncle.

CHAPTER 2

Installation

At the command line:

```
$ easy_install pyramid_transactional_celery
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv pyramid_transactional_celery
$ pip install pyramid_transactional_celery
```

CHAPTER 3

Usage

To use Transactional Celery for Pyramid in a project:

```
import pyramid_transactional_celery
```

CHAPTER 4

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:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/petrilli/pyramid_transactional_celery/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.

4.1.2 Fix Bugs

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

4.1.3 Implement Features

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

4.1.4 Write Documentation

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

4.1.5 Submit Feedback

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

4.2 Get Started!

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

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

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

4.3 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 2.6, 2.7, 3.3, and 3.4, and for PyPy. Check https://travis-ci.org/petrilli/pyramid_transactional_celery/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

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

CHAPTER 5

Credits

5.1 Development Lead

- Christopher Petrilli <petrilli@amber.org>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

CHAPTER 7

0.1.1 (2015-01-19)

- Removed an excess creation of a CeleryDataManager that was a left-over from a previous approach. While this didn't create a bug, it wasted memory.

CHAPTER 8

0.1.0 (2015-01-19)

- Initial functionality, but more testing of edge cases is needed to ensure that it works correctly in all cases, and with other versions of Celery.
- First release on PyPI.

CHAPTER 9

Indices and tables

- *genindex*
- *modindex*
- *search*