Flask-HashFS Documentation

Release 0.3.0

Derrick Gilland

Sep 27, 2017

Contents

Flask extension for HashFS, a content-addressable file management system.

CHAPTER 1

What is HashFS?

HashFS is a content-addressable file management system. What does that mean? Simply, that HashFS manages a directory where files are saved based on the file's hash.

Typical use cases for this kind of system are ones where:

- Files are written once and never change (e.g. image storage).
- It's desirable to have no duplicate files (e.g. user uploads).
- File metadata is stored elsewhere (e.g. in a database).

CHAPTER 2

What is Flask-HashFS?

Flask-HashFS is a Flask extension that integrates HashFS into the Flask ecosystem.

CHAPTER 3

Links

- Project: https://github.com/dgilland/flask-hashfs
- Documentation: http://flask-hashfs.readthedocs.org
- PyPI: https://pypi.python.org/pypi/flask-hashfs/
- TravisCI: https://travis-ci.org/dgilland/flask-hashfs

CHAPTER 4

Quickstart

Install using pip:

```
pip install Flask-HashFS
```

Initialization

```
from flask import Flask
from flask_hashfs import FlaskHashFS

app = Flask(__name__)
fs = FlaskHashFS()
```

Configure `Flask-HashFS` to store files in `/var/www/data/uploads` and give them a route prefix at `/uploads`.

```
app.config.update({
    'HASHFS_HOST': None,
    'HASHFS_PATH_PREFIX': '/uploads',
    'HASHFS_ROOT_FOLDER': '/var/www/data/uploads',
    'HASHFS_DEPTH': 4,
    'HASHFS_WIDTH': 1,
    'HASHFS_ALGORITHM': 'sha256'
})

fs.init_app(app)
```

Usage

Use Flask-HashFS to manage files using HashFS.

```
with app.app_context():
    # Store readable objects or file paths
    address = fs.put(io_obj, extension='.jpg')


    # Get a file's hash address
    assert fs.get(address.id) == address
    assert fs.get(address.relpath) == address
    assert fs.get(address.abspath) == address
    assert fs.get('invalid') is None


    # Get a BufferedReader handler
    fileio = fs.open(address.id)

    # Or using the full path...
    fileio = fs.open(address.abspath)

    # Or using a path relative to fs.root
    fileio = fs.open(address.relpath)


    # Delete a file by address ID or path
    fs.delete(address.id)
    fs.delete(address.abspath)
    fs.delete(address.relpath)
```

For direct access to the HashFS instance, use the `client` attribute.

```
fs.client
assert isinstance(fs.client, flask_hashfs.HashFS)
```

Generate URLs for HashFS content.

```
with app.test_request_context():
    fs.url_for('relative/file/path')
```

For more details, please see the full documentation at http://flask-hashfs.readthedocs.org and http://hashfs.readthedocs.org.

CHAPTER 5

Guide

Installation

flask-hashfs requires Python >= 2.7 or >= 3.3.

To install from PyPI:

```
pip install flask-hashfs
```

API Reference

The flask-hashfs module.

Flask extension for HashFS, a content-addressable file management system.

class `flask_hashfs.`**`FlaskHashFS`**(*app=None*)

Flask extension for storing files on file system using hashfs.

Configuration values:

`HASHFS_HOST`	Host where files are served. Set if files are served from a different host than application. Defaults to `None` which uses `flask.request.host_url`.
`HASHFS_PATH_PREFIX`	URL path prefix where files are served. Defaults to `''`.
`HASHFS_ROOT_FOLDER`	Root folder to save files. Must be set.
`HASHFS_DEPTH`	Number of nested folders to use when saving files. Defaults to `4`.
`HASHFS_WIDTH`	Width of each nested subfolder. Defaults to `1`.
`HASHFS_ALGORITHM`	Hashing algorithm to use when computing content hash. Defaults to `'sha256'`.

`client`
Underlying `HashFS` instance.

`url_for`(*relpath*, *external=True*)
Return URL for path relative to `HASHFS_ROOT_FOLDER`.

Parameters

- **`relpath`** (*`str`*) – Relative path to `HASHFS_ROOT_FOLDER` where file is located.
- **`external`** (*`bool`*) – Whether to include host in URL.

Returns URL for path.

Return type str

Note: This function builds the URL with the assumption that *relpath* is a valid file path. It does not check for file existence.

class `flask_hashfs.`**`HashAddress`**
File address containing file's path on disk and it's content hash ID.

`id`
str – Hash ID (hexdigest) of file contents.

`relpath`
str – Relative path location to `HashFS.root`.

`abspath`
str – Absoluate path location of file on disk.

`is_duplicate`
boolean, optional – Whether the hash address created was a duplicate of a previously existing file. Can only be `True` after a put operation. Defaults to `False`.

CHAPTER 6

Project Info

License

The MIT License (MIT)

Copyright (c) 2015, Derrick Gilland

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Versioning

This project follows Semantic Versioning with the following caveats:

- Only the public API (i.e. the objects imported into the flask-hashfs module) will maintain backwards compatibility between MINOR version bumps.
- Objects within any other parts of the library are not guaranteed to not break between MINOR version bumps.

With that in mind, it is recommended to only use or import objects from the main module, flask-hashfs.

Changelog

v0.3.0 (2015-06-03)

- Replace manual proxy access of `HashFS` methods with single `__getattr__` method.

v0.2.0 (2015-06-02)

- Pin `hashfs` dependency to `>=0.3.0`. (**breaking change**)
- Rename config key `HASHFS_LENGTH` to `HASHFS_WIDTH` to be in alignment with `hashfs>=0.3.0`. (**breaking change**)

v0.1.0 (2015-06-02)

- First release.
- Add `FlaskHashFS.put`.
- Add `FlaskHashFS.get`.
- Add `FlaskHashFS.open`.
- Add `FlaskHashFS.delete`.
- Add `FlaskHashFS.url_for`.

Authors

Lead

- Derrick Gilland, dgilland@gmail.com, dgilland@github

Contributors

None

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:

Types of Contributions

Report Bugs

Report bugs at https://github.com/dgilland/flask-hashfs/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 “enhancement” or “help wanted” is open to whoever wants to implement it.

Write Documentation

Flask-HashFS could always use more documentation, whether as part of the official Flask-HashFS 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/dgilland/flask-hashfs/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 :)

Get Started!

Ready to contribute? Here’s how to set up `flask-hashfs` for local development.

1. Fork the `flask-hashfs` repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/flask-hashfs.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenv installed, this is how you set up your fork for local development:

   ```
   $ cd flask-hashfs
   $ make build
   ```

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 linting (PEP8 and pylint) and the tests, including testing other Python versions with tox:

   ```
   $ make test-full
   ```

6. Add yourself to `AUTHORS.rst`.
7. 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
   ```

8. Submit a pull request through the GitHub website.

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 README.rst.
3. The pull request should work for Python 2.7, 3.3, and 3.4. Check https://travis-ci.org/dgilland/flask-hashfs/pull_requests and make sure that the tests pass for all supported Python versions.

Project CLI

Some useful CLI commands when working on the project are below. **NOTE:** All commands are run from the root of the project and require `make`.

make build

Run the `clean` and `install` commands.

```
make build
```

make install

Install Python dependencies into virtualenv located at `env/`.

```
make install
```

make clean

Remove build/test related temporary files like `env/`, `.tox`, `.coverage`, and `__pycache__`.

```
make clean
```

make test

Run unittests under the virtualenv's default Python version. Does not test all support Python versions. To test all supported versions, see *make test-full*.

```
make test
```

make test-full

Run unittest and linting for all supported Python versions. **NOTE:** This will fail if you do not have all Python versions installed on your system. If you are on an Ubuntu based system, the Dead Snakes PPA is a good resource for easily installing multiple Python versions. If for whatever reason you're unable to have all Python versions on your development machine, note that Travis-CI will run full integration tests on all pull requests.

```
make test-full
```

make lint

Run `make pylint` and `make pep8` commands.

```
make lint
```

make pylint

Run `pylint` compliance check on code base.

```
make pylint
```

make pep8

Run PEP8 compliance check on code base.

```
make pep8
```

make docs

Build documentation to `docs/_build/`.

```
make docs
```

CHAPTER 7

Indices and Tables

- genindex
- modindex
- search

Python Module Index

Index