dataflake.wsgi.cheroot

Apr 20, 2019

Contents

1 Installation **3**

2 Using this package **5**
- 2.1 Using the PasteDeploy entry point . . . 5
- 2.2 Creating a basic WSGI configuration for Zope . . . 6

3 Development **7**
- 3.1 Getting the source code . . . 7
- 3.2 Setting up a development sandbox and testing . . . 7
- 3.3 Building the documentation . . . 7
- 3.4 Making a release . . . 8

4 Changelog **9**
- 4.1 1.1 (unreleased) . . . 9
- 4.2 1.0 (2019-04-20) . . . 9

5 Indices and tables **11**

This package provides a PasteDeploy-compatible entry point to easily integrate the cheroot WSGI server into an environment that uses PasteDeploy-style `.ini` files to compose a WSGI application.

It also includes a script to create a basic WSGI configuration file for Zope, similar to Zope's own `mkwsgiinstance`, but specifying `cheroot` instead of `waitress` as WSGI server.

CHAPTER 1

Installation

You will need Python 2.7 or 3.5 and higher to install and use `dataflake.wsgi.cheroot`.

It is advisable to install `dataflake.wsgi.cheroot` into a virtualenv or similar environment like a buildout from `zc.buildout` to obtain isolation from any "system" packages you've got installed in your Python version (and likewise, to prevent `dataflake.wsgi.cheroot` from globally installing versions of packages that are not compatible with your system Python).

Setuptools/Distutils:

```
$ easy_install dataflake.wsgi.cheroot
```

Pip:

```
$ pip install dataflake.wsgi.cheroot
```

If you use `zc.buildout` you can add `dataflake.wsgi.cheroot` to the necessary `eggs` section to have it pulled in automatically:

```
...
eggs =
    dataflake.wsgi.cheroot
...
```

CHAPTER 2

Using this package

2.1 Using the PasteDeploy entry point

You can use the PasteDeploy entry point in your WSGI configuration file to define a `cheroot` server:

```
[server:main]
use = egg:dataflake.wsgi.cheroot#main
host = 127.0.0.1
port = 8080
```

If you leave out the `host` specification, `cheroot` will listen on all IPv4 interfaces (*0.0.0.0*). The default port, if none is given, is 8080.

`cheroot` supports a wide range of configuration options that you can pass as part of your WSGI configuration. Here's an example showing all options:

```
[server:main]
use = egg:dataflake.wsgi.cheroot#main
host = 127.0.0.1
port = 8080
server_name = MyServer
max = -1
request_queue_size = 5
timeout = 10
shutdown_timeout = 5
accepted_queue_size = -1
accepted_queue_timeout = 10
peercreds_enabled = False
peercreds_resolve_enabled = False
```

The possible options are listed in the cheroot documentation, but a more detailed explanation is in the code itself, look for the definition of the `Server` class `__init__` method.

2.2 Creating a basic WSGI configuration for Zope

This package defines a console script named `mkcherootinstance` that works just like Zope's own `mkwsgiinstance`. It will ask you for a location, a username and a password to create a basic Zope instance home with a WSGI configuration, in this case it will be `cheroot`-based as opposed to Zope's default, `waitress`.

Note: Just like `mkwsgiinstance`, the script will not overwrite an existing WSGI configuration file at `etc/zope.ini`. You need to move the existing file to the side to get a fresh configuration.

```
$ bin/mkcherootinstance
Please choose a directory in which you'd like to install
Zope "instance home" files such as database files, configuration
files, etc.

Directory: .
Please choose a username and password for the initial user.
These will be the credentials you use to initially manage
your new Zope instance.

Username: admin
Password: (enter password)
Verify password: (re-enter password)
```

CHAPTER 3

Development

3.1 Getting the source code

The source code is maintained on GitHub:

```
$ git clone https://github.com/dataflake/dataflake.wsgi.cheroot.git
```

3.2 Setting up a development sandbox and testing

Once you've obtained a source checkout, you can follow these instructions to perform various development tasks. All development requires that you run the buildout from the package root directory:

```
$ python2.7 bootstrap.py
$ bin/buildout
```

Once you have a buildout, the tests can be run as follows:

```
$ bin/test
```

3.3 Building the documentation

The Sphinx documentation is built by doing the following from the directory containing setup.py:

```
$ cd docs
$ make html
```

The finished HTML files are under *docs/_build/html*.

3.4 Making a release

These instructions assume that you have a development sandbox set up using `zc.buildout` as the scripts used here are generated by the buildout:

```
$ bin/buildout -N
$ bin/buildout setup setup.py sdist bdist_wheel
$ bin/twine upload -s dist/*
```

The `bin/buildout` step will make sure the correct package information is generated.

CHAPTER 4

Changelog

4.1 1.1 (unreleased)

4.2 1.0 (2019-04-20)

- Initial release

CHAPTER 5

Indices and tables

- genindex
- search
- glossary