dataflake.wsgi.werkzeug

Apr 24, 2019

Contents

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

A second entry point will enable the `werkzeug` debugger, so you get nice clickable tracebacks with the ability to open a console prompt at any point in the stack. The debugger is explained in the werkzeug documentation and **should never be running in production**.

It also includes a script to create a basic WSGI configuration file for Zope, similar to Zope's own `mkwsgiinstance`, but specifying `werkzeug` 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.werkzeug`.

It is advisable to install `dataflake.wsgi.werkzeug` 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.werkzeug` from globally installing versions of packages that are not compatible with your system Python).

Setuptools/Distutils:

```
$ easy_install dataflake.wsgi.werkzeug
```

Pip:

```
$ pip install dataflake.wsgi.werkzeug
```

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

```
...
eggs =
    dataflake.wsgi.werkzeug
...
```

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 `werkzeug` server:

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

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

`werkzeug` supports a wide range of configuration options that you can pass as part of your WSGI configuration. They are listed in the werkzeug documentation.

2.2 Creating a basic WSGI configuration for Zope

This package defines a console script named `mkwerkzeuginstance` 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 `werkzeug`-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/mkwerkzeuginstance
Please choose a directory in which you'd like to install
Zope "instance home" files such as database files, configuration
files, etc.

```

(continues on next page)

(continued from previous page)

```
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)
```

2.3 Using the werkzeug debugger

This package has a second entry point to enable the werkzeug debugger.

Warning: Do not enable the werkzeug debugger in production! Any site visitor may execute Python code on your server using the debugger console!

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

In the `[server]` section, you can combine the werkzeug server options and the werkzeug debugger options

The debugger will present you with a nice exception traceback display in the browser and the ability to open a console prompt at any point in the traceback call stack. This is great for developers, but **dangerous if exposed to the wider Internet**, so never leave this enabled on a production site.

CHAPTER 3

Development

3.1 Getting the source code

The source code is maintained on GitHub:

```
$ git clone https://github.com/dataflake/dataflake.wsgi.werkzeug.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 (2019-04-24)

- Initial release

CHAPTER 5

Indices and tables

- genindex
- search
- glossary