pypicache Documentation

Release 0.1

Michael Twomey

April 02, 2014

pypicache aims to solve some problems developers and teams encounter when using python packages:

1. Many python package installation tools will check all associated links for a python project on PyPI, which can be problematic when the project's server is down. Doubly so if the download link is on that server.
2. Commercial development of python projects might involve local patches to packages or completely private packages. It's useful to host these internally.
3. Hosting an internal proxy can save quite a bit of bandwidth, which might be an issue for some teams.
4. Installation of a larger project can be noticably faster from an internal server.
5. Continuous integration tools can potentially install large sets of packages again and again, which can consume upstream bandwidth and slow down builds.

pypicache can be used in the following ways:

1. As a straight proxy to PyPI, caching package downloads where possible.
2. As a completely standalone PyPI server, useful for deploying from.
3. As an internal server for hosting custom packages.

A possible day to day workflow could involve a pypicache server running on developer's machines or in an office. Developers would install packages via this server. This server can also be shared by a deployment build tool which would install from the completely local copy of packages. This allows for repeatable builds.

CHAPTER 1

Installation

Currently this is only available as a development project in bitbucket: https://bitbucket.org/micktwomey/pypicache/

To install:

```
pip install -e git+git@bitbucket.org:micktwomey/pypicache.git#egg=pypicache
cd src/pypicache
pip install -r requirements.txt
```

CHAPTER 2

Usage

Running the server is fairly straightforward:

```
python -m pypicache.main /tmp/mypackages
```

This will fire up the server with a cache in /tmp/mypackages.

You can start using the server with normal tools as a proxy:

```
pip install -i http://localhost:8080/simple somepackage
```

Installation should proceed normally, and the package should appear inside /tmp/mypackages.

You can then install the package from a completely local cache, without hitting any external servers using:

```
pip install -i http://localhost:8080/local somepackage
```

If you want to take a requirements.txt file and cache the packages it specifies you can POST the file:

```
pip freeze | curl -X POST -F requirements=@- http://localhost:8080/requirements.txt | python -m json
```

or:

```
curl -X POST -F requirements=@requirements.txt http://localhost:8080/requirements.txt | python -m jso
```

You can also upload packages directly, either into the normal PyPI package location via a POST:

```
curl -X POST -F sdist=@dist/mypackage-1.0.tar.gz  http://localhost:8080/uploadpackage/
```

CHAPTER 3

URLs

An enumeration of the current api:

- GET /
- GET /simple/mypackage
- **GET /local/mypackage**
  - Currently case sensitive
- POST /requirements.txt
- POST /uploadpackage/ - Applies simple logic to parse package name - Can't overwrite packages
- GET /packages/source/m/mypackage/mypackage-1.0.tar.gz - Checks PyPI if not present locally
- GET /packages/2.7/m/mypackage/mypackage-1.0-py2.7.egg - not implemented
- PUT /packages/2.7/m/mypackage/mypackage-1.0-py2.7.egg - not implemented