Flask-Pypi-Proxy Documentation

Release 0.1.0

Tomas Zulberti

July 31, 2015

Contents

1 Introduction **3**
1.1 Advantages . . . 3
1.2 Disadvantages . . . 3

2 Deploying **5**
2.1 Configuration . . . 5
2.2 Debian/Ubuntu example installation . . . 6
2.3 More advanced configuration . . . 7

3 Using the proxy **9**
3.1 Downloading packages . . . 9
3.2 Uploading packages . . . 9

4 Indices and tables **11**

Contents:

CHAPTER 1

Introduction

Flask-Pypi-Proxy works is a proxy for PyPI that also enables you to upload your custom packages.

1.1 Advantages

- Once a package is downloaded from PyPI, then it won't be downloaded again. Bacause of this, the package installation is quicker. Let's assume that you have some servers where your application run. You configure one of these servers with Flask-Pypi-Proxy, and because of that all the servers download the required Python packages from an internal server.
- You can upload your custom eggs. For example, you have your project which is close sourced and because of that it can't be uploaded to PyPI. This will solve the problem because you will have an internal package system.
- Upload compiled packages. Some packages (lxml, Pillow) compile using the system libraries. If all the servers are using the same versions of the libraries, then you can upload the compiled package and save the compilation time for each install.

1.2 Disadvantages

- When downloading the package for the first time, it will take some extra time. This happens because the package will have to be downloaded from PyPI first, and then from the Flask-Pypi-Proxy.

CHAPTER 2

Deploying

Flask-Pypi-Proxy is a normal Python application, so it can be deployed as any other Flask application. For more information, you can check here: http://flask.pocoo.org/docs/deploying/

2.1 Configuration

The project uses the environment key **FLASK_PYPI_PROXY_CONFIG** that references the path where the configuration file is. This file is in JSON format with the following keys:

BASE_FOLDER_PATH the base path where all the eggs will be stored. For example: /home/pypi/eggs. For each egg name, a subfolder will be created inside this folder. This value is required.

PRIVATE_EGGS a list with all the private eggs. For these eggs, PyPI won't be used and it will only answer with local data. By default, there are no private eggs, so everytime it will hit PyPI. This is useful for private projects, or for eggs that were uploaded after compilation (lxml, PIL, etc...)

PYPI_URL the URL where from where the eggs should be downloaded. By default it uses: http://pypi.python.org but it can be changed to any other. This might be useful, for example if you have a development/local proxy, that uses the production proxy.

LOGGING_PATH the complete filepath of the logging for the application. This file must include the path and the filename. This value is required.

LOGGING_LEVEL the string that represents the logging level that the application should use. In this case, it should be a string: DEBUG, INFO, WARNING, ERROR. By default, it uses DEBUG.

SHOULD_USE_EXISTING if True, then when getting the index for a package, ie the different versions, it will use the ones that exists on the local repo instead of using the information found on PYPI_URL. If False, and the package isn't private, then it will use the PYPI_URL to get the file package. Setting this value to True might do the things a little faster, but on the other hand, it might get you into trouble if trying to download a version of a package that doesn't exist, but other versions do exist. For example, you download version 1.4.2 of Django. If this value is set to True, then if you try to download version 1.5, it will return that it doesn't exist.

If you don't want to use a configuration file, then environment variables could be used. The variable names are the same as the keys in the configuration file, but they take the prefix: **PYPI_PROXY**. So the variable names used are:

- PYPI_PROXY_BASE_FOLDER_PATH
- PYPI_PROXY_LOGGING_PATH
- PYPI_PROXY_LOGGING_LEVEL
- PYPI_PROXY_PRIVATE_EGGS
- PYPI_PROXY_PYPI_URL

- PYPI_PROXY_SHOULD_USE_EXISTING

If the configuration file exists, then the values that are in the file will be used and not the values of the system environment.

An example of the file could be:

```
{
    BASE_FOLDER_PATH: '/mnt/eggs/',
    LOGGING_PATH: '/mnt/eggs/debug.log',
    PRIVATE_EGGS: [
        'miproject1',
        'miproject2',
        'miproject3'
        ],
    PYPI_URL: 'https://pypy.miserver.com'
}
```

But a more common configuration file, will be:

```
{
    BASE_FOLDER_PATH: '/mnt/eggs/',
    LOGGING_PATH: '/mnt/eggs/debug.log',
}
```

2.2 Debian/Ubuntu example installation

This is a **VERY** simple/basic configuration. It doesn't provide any authentication, so this shouldn't be used on production.

```
>>> sudo apt-get install apache2 libapache2-mod-wsgi
>>> sudo apt-get install python-setuptools python-dev libxml2-dev libxslt-dev
>>> sudo easy_install Flask-Pypi-Proxy

>>> mkdir -p /mnt/eggs/
>>> sudo chown www-data:www-data -R /mnt/eggs/
```

Now, lets create the WSGI configuration file (in this example, I will create it on /mnt/eggs/flask_pypi_proxy.wsgi). The content of that file will be something like:

```
import os

os.environ['PYPI_PROXY_BASE_FOLDER_PATH'] = '/mnt/eggs/'
os.environ['PYPI_PROXY_LOGGING_PATH'] = '/mnt/eggs/proxy.logs'

# if installed inside a virtualenv, then do this:
# activate_this = 'VIRTUALENENV_PATH/bin/activate_this.py'
# execfile(activate_this, dict(__file__=activate_this))

from flask_pypi_proxy.views import app as application
```

Finally, the Apache configuration. Create a file at /etc/apache2/sites-enabled/flask_pypi_proxy with the following content:

```
<VirtualHost *:80>
    WSGIDaemonProcess pypi_proxy threads=5
    WSGIScriptAlias / /mnt/eggs/flask_pypi_proxy.wsgi
</VirtualHost>
```

Restart Apache

```
>>> sudo service apache2 restart
```

2.3 More advanced configuration

The following steps will show you how to install this service inside a virtualenv, also using HTTP basic auth to create some security for the eggs.

```
>>> sudo apt-get install apache2 libapache2-mod-wsgi
>>> sudo apt-get install python-setuptools python-dev libxml2-dev libxslt-dev
```

Now, create the user where the virtualenv will be installed:

```
>>> sudo adduser pypi-proxy
Adding user `pypi-proxy' ...
Adding new group `pypi-proxy' (1001) ...
Adding new user `pypi-proxy' (1001) with group `pypi-proxy' ...
Creating home directory `/home/pypi-proxy' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
>>> sudo easy_install virtualenv
>>> sudo su - pypi-proxy
```

The following steps will be executed as **pypi-proxy**:

```
mkdir ~/envs
virtualenv ~/envs/proxy
source ~/envs/proxy/bin/activate
pip instal Flask-Pypi-Proxy
mkdir /home/pypi-proxy/eggs/ # where the eggs will be
chgrp www-data /home/pypi-proxy/eggs/
chmod 775 /home/pypi-proxy/eggs/
mkdir /home/pypi-proxy/logs/ # the same but for the logs files
chgrp www-data /home/pypi-proxy/logs/
chmod 775 /home/pypi-proxy/logs/

htpasswd -c /home/pypi-proxy/htpasswd.file MY_USERNAME # creates the password file
sudo chown www-data:www-data /home/pypi-proxy/htpasswd.file
sudo chmod 620 /home/pypi-proxy/htpasswd.file
```

Under the same user, lets create the WSGI file (for this example, I will put it on /home/pypi-proxy/pypi-proxy.wsgi). The content of this file is as follows:

```
import os

os.environ['PYPI_PROXY_BASE_FOLDER_PATH'] = '/home/pypi-proxy/eggs/'
os.environ['PYPI_PROXY_LOGGING_PATH'] = '/home/pypi-proxy/logs/proxy.log'

# if installed inside a virtualenv, then do this:
activate_this = '/home/pypi-proxy/envs/proxy/bin/activate_this.py'
execfile(activate_this, dict(__file__=activate_this))

from flask_pypi_proxy.views import app as application
```

Now return to the normal user, and create the following Apache configuration (/etc/apache2/sites-enabled/flask_pypi_proxy):

```
<VirtualHost *:80>
    <Location />
    AuthType Basic
    AuthUserFile /home/pypi-proxy/htpasswd.file
    AuthName "Private files"
    Require valid-user
    Order deny,allow
    Allow from all
    </Location>

    WSGIDaemonProcess pypi_proxy threads=5
    WSGIScriptAlias / /home/pypi-proxy/proxy.wsgi
</VirtualHost>
```

Restart Apache

```
sudo service apache2 restart
```

CHAPTER 3

Using the proxy

3.1 Downloading packages

To download a package from the proxy, there are two choices:

- Specify the server where the server is installed when runing **pip** or **easy_install**.

```
pip install -i http://mypypiproxy/simple/ Flask
easy_install -i http://mypypiproxy/simple/ Flask
```

- Use the index url in a configuration file. For easy_install, it should be on **~/.pydistutils.cfg** (on Linux), and the file should have the following format:

```
[easy_install]
index_url = http://mypypiproxy/simple/
```

For **pip**, the configuration file is **~/.pip/pip.conf**, and the file should have the following format:

```
[global]
index-url = http://mypypiproxy/simple/
```

Also, you should increment the timeout option for **pip** or **easy_install**. For pip, the **~/.pip/pip.conf** configuration file should be something like:

[global] index-url = http://mypypiproxy/simple/ timeout = 60

3.2 Uploading packages

To upload a package, the **~/.pypirc** should be updated to something like:

```
[distutils]
index-servers =
    miserver

[myserver]
username:foo
password:bar
repository:http://mypypiproxy/pypi/
```

I you are using the configuration with basic auth, then the configuration file should look something like this:

Basically, you should put in the username and password used for the basic auth.

The username and password values aren't required by Flask-Pypi-Proxy. They are used by distutils when uploading the package. If you don't have any authentication after this, then you can put any values. After that, go to the **setup.py** of your project and run:

```
python setup.py sdist upload -r myserver
```

IMPORTANT: The command *register*, won't work if you are using basic auth. For example, if you run

```
python setup.py register
```

and if your server is configured using basic auth, then register will return a 401 error. Simply upload the package without running register.

Indices and tables

- genindex
- modindex
- search