Localshop Documentation

Release 0.8.3

Michael van Tellingen

March 01, 2017

Contents

Localshop is a PyPI server which automatically proxies and mirrors PyPI packages based upon packages requested. It also supports the uploading of local (private) packages.

Contents:

CHAPTER 1

Installing

Download and install localshop via the following command:

```
pip install localshop
```

This should best be done in a new virtualenv. Now initialize your localshop environment by issuing the following command:

```
localshop init
```

If you are upgrading from an earlier version simply run:

```
localshop upgrade
```

And then start it via:

```
gunicorn localshop.wsgi:application
```

You will also need to start the celery daemon, it's responsible for downloading and updating the packages from PyPI. So open another terminal, activate your virtualenv (if you have created one) and run the following command:

```
localshop celery worker -B -E
```

You can now visit http://localhost:8000/ and view all the packages in your localshop!

Note: If you prefer to start listening on a different network interface and HTTP port, you have the pass the parameter `-b` to `gunicorn`. For example, the following command starts localshop on port 7000 instead of 8000:

```
gunicorn localshop.wsgi:application -b 0.0.0.0:7000
```

The next step is to give access to various hosts to use the shop. This is done via the webinterface (menu -> permissions -> cidr). Each ip address listed there will be able to download and upload packages. If you are unsure about ips configuration, but still want to use authentication, specify "0.0.0.0/0" as the unique cidr configuration. It will enable for any ip address.

Docker alternative

Install docker and docker-compose and then run:

```
cp docker.conf.py{.example,}
docker-compose build
docker-compose run localshop syncdb
docker-compose run localshop createsuperuser
docker-compose up
```

You should be able to see localshop running in *http://docker-host:8000*.

CHAPTER 2

How it works

Packages which are requested and are unknown are looked up on pypi via the xmlrpc interface. At the moment the client downloads one of the files which is not yet mirror'ed a 302 redirect is issued to the correct file (on pypi). At that point the worker starts downloading the package and stores it in ~/.localshop/files so that the next time the package is request it is available within your own shop!

Uploading local/private packages

To upload your own packages to your shop you need to modify/create a .pypirc file. See the following example:

```
[distutils]
index-servers =
    local

[local]
username: myusername
password: mysecret
repository: http://localhost:8000/simple/
```

To upload a custom package issue the following command in your package:

```
python setup.py upload -r local
```

It should now be available via the webinterace

Using the shop for package installation

To install packages with pip from your localshop add *-i* flag, e.g.:

```
pip install -i http://localhost:8000/simple/ localshop
```

or edit/create a ~/.pip/pip.conf file following this template:

```
[global]
index-url = http://<access_key>:<secret_key>@localhost:8000/simple
```

Then just use pip install as you are used to do. You can replace access_key and secret_key by a valid username and password.

Credentials for authentication

If you don’t want to use your Django username/password to authenticate uploads and downloads you can easily create one of the random credentials localshop can create for you.

Go to the Credentials section and click on create. Use the access key as the username and the secret key as the password when uloading packages. A `~/.pypirc` could look like this:

```
[distutils]
index-servers =
    local

[local]
username: 4baf221849c84a20b77a6f2d539c3e8a
password: 200984e70f0c463b994388c4da26ec3f
repository: http://localhost:8000/simple/
```

pip allows you do use those values in the index URL during download, e.g.:

```
pip install -i http://<access_key>:<secret_key>@localhost:8000/simple/ localshop
```

So for example:

```
pip install -i http://4baf221849c84a20b77a6f2d539c3e8a:200984e70f0c463b994388c4da26ec3f@localhost:80
```

Warning: Please be aware that those credentials are transmitted unencrypted over http unless you setup your localshop instance to run on a server that serves pages via https.

In case you ever think a credential has been compromised you can disable it or delete it on the credential page.

CHAPTER 3

Adding users

You can add users using the Django admin backend at `/admin`. In order for the user to be able to generate credentials for his account, he needs the following four user permissions:

- `permissions.add_credential`
- `permissions.change_credential`
- `permissions.delete_credential`
- `permissions.view_credential`

CHAPTER 4

Settings

There are a few settings to set in `~/.localshop/localshop.conf.py` that change the behaviour of the localshop.

LOCALSHOP_DELETE_FILES

default `False`

If set to `True` files will be cleaned up after deleting a package or release from the localshop.

LOCALSHOP_DISTRIBUTION_STORAGE

default `'storages.backends.overwrite.OverwriteStorage'`

The dotted import path of a Django storage class to be used when uploading a release file or retrieving it from PyPI.

LOCALSHOP_HTTP_PROXY

default `None`

Proxy configuration used for Internet access. Expects a dictionary configured as mentioned by http://docs.python-requests.org/en/latest/user/advanced/#proxies

LOCALSHOP_ISOLATED

default `False`

If set to `True` Localshop never will try to redirect the client to PyPI. This is useful for environments where the client has no Internet connection.

Note: If you set `LOCALSHOP_ISOLATED` to `True`, client request can be delayed for a long time because the package must be downloaded from Internet before it is served. You may want to set pip environment variable `PIP_DEFAULT_TIMEOUT` to a big value. Ex: `300`

`LOCALSHOP_USE_PROXIED_IP`

default `False`

If set to `True` Localshop will use the X-Forwarded-For header to validate the client IP address. Use this when Localshop is running behind a reverse proxy such as Nginx or Apache and you want to use IP-based permissions.

`LOCALSHOP_RELEASE_OVERWRITE`

default `True`

If set to `False`, users will be preveneted from overwriting already existing release files. Can be used to encourage developers to bump versions rather than overwriting. This is PyPI's behaviour.

`LOCALSHOP_VERSIONING_TYPE`

default `None`

If set to `False`, no versioning "style" will be enforced.

If you want to validated versions you can choose any Versio available backends.

IMPORTANT the value of this config must be a full path of the wanted class e.g. *versio.version_scheme.Pep440VersionScheme*.

- **Simple3VersionScheme** which supports 3 numerical part versions (A.B.C where A, B, and C are integers)
- **Simple4VersionScheme** which supports 4 numerical part versions (A.B.C.D where A, B, C, and D are integers)
- **Pep440VersionScheme** which supports PEP 440 versions (N[.N]+[{a|b|c|rc}N][.postN][.- devN][+local])
- **PerlVersionScheme** which supports 2 numerical part versions where the second part is at least two digits A.BB where A and B - are integers and B is zero padded on the left. For example: 1.02, 1.34, 1.567)

CHAPTER 5

Contributing

Want to contribute with Localshop? Great! We really appreciate your help. But before digging into your new fluffy-next-millionaine-feature code keep in mind that you **MUST** follow this guide to get your pull requests approved.

Get started

1. Fork the project and follow the installation instructions [1].
2. Your code **MUST** contain tests. This is a requirement and no pull request will be approved if it lacks tests. Even if your're making a small bug fix we want to ensure that it will not introduce any another bug.
3. Help to keep the documentation up-to-date is really appreaciated. Always check if your're making changes that make the documentation obsolete and update it.
4. Squash your commits before making a pull request whenever possible. This will avoid history pollution with middle commits that breaks things. Your pull request should be a single commit with all your changes.
5. Open a pull request. Usually, the target branch at the main repository will be `develop`, but if your're sending a bugfix to avoid the extinction of human race, maybe you want to target the `master` branch.

Tip: Use a meaningful and convincing pull request description. Feel free to use emojis to give us a clue of what kind changes your're making. The *Style guide* contains some of our preferred ones.

Style guide

- Follow the PEP8. The only exception is the maximum line width. We uses 120 characters instead of 79.
- Make sure that your code does not raises any Pylint errors or warnings.
- Always group the imports in 3 blocks: native libraries, third party libraries and project imports.
- Keep the import block alphabetically ordered. If you use Sublime Text, you can do this by selecting the import block and hitting `F9`
- Avoid polluting the current namespace with lots of imports. If you find yourself in a situation of importing a lot of symbols from the same package, consider import the package itself.

  Wrong way:

[1] *Installing*

```
from django.core.exceptions import (ImproperlyConfigured, AppRegistryNotReady, FieldError, Disal
                                    DisallowedRedirect, DjangoRuntimeWarning)
```

Preferred way:

```
from django.core import exceptions as djexc
```

Commit messages

- Limit the first line to 72 characters or less
- Always use English
- **Consider starting the commit message with an applicable emoji:**
  - `:lipstick:` when improving the format/structure of the code
  - `:fire:` when removing code or files
  - `:bug:` when fixing a bug
  - `:beetle:` when fixing a bug
  - `:book:` when writing docs
  - `:green_heart:` when fixing the CI build
  - `:white_check_mark:` when adding tests
  - `:x:` when commiting code with failed tests
  - `:arrow_up:` when upgrading dependencies
  - `:arrow_down:` when downgrading dependencies

CHAPTER 6

Indices and tables

- genindex
- modindex
- search