extrapypi Documentation

Release

karec

Feb 02, 2018

Contents

External pypi server with web ui and basics permissions management extrapypi don't mirror official pypi packages, and will not. It's just not build with this goal in mind. Extrapypi is just here to provide you an extra index to upload and install private packages from your own index.

CHAPTER 1

Features

- Upload packages from twine / setuptools
- Install packages with pip using only extra-index option
- Basics permissions management using roles (currently admin, developer, maitainer, installer)
- Easy deployment / installation using the WSGI server you want
- MySQL, PostgresSQL and SQLite support
- Extensible storage system
- CLI tools to help you deploy / init / test extrapypi
- Basic dashboard to visualize packages and users

CHAPTER 2

Documentation

2.1 Installation

2.1.1 Using pip

Recommanded way to install extrapypi is to use the latest version hosted on PyPI :

```
pip install extrapypi
```

2.1.2 Installing via git

Extrapypi is hosted at https://github.com/karec/extrapypi, you can install it like this

```
git clone https://github.com/karec/extrapypi.git
cd extrapypi
python setup.py install
```

Or, for development

```
pip install -e .
```

2.1.3 Starting and init extrapypi

Once installed, you will need to create a configuration file and create database, tables and users. But don't worry, we provide commands in CLI tools to do that.

First thing we need : generate a configuration file

```
extrapypi start --filename myconfig.py
```

This will generate a minimal configuration file, full documentation on configuration is avaible in the *configuration section*.

Once configuration done, you can create database, tables and default users using init command

```
EXTRAPYPI_CONFIG=/path/to/myconfig.cfg extrapypi init
```

It will create two users :

- User admin, with password admin and role admin
- User pip with password pip and role installer

After that you can start extrapypi using run command

```
EXTRAPYPI_CONFIG=/path/to/myconfig.cfg extrapypi run
```

Warning: You will need proper database drivers if you want to use anything else than sqlite. See http://docs.sqlalchemy.org/en/latest/dialects/ for more informations about avaible dialects

2.2 Extrapypi configuration

All settings present here can be override with your own configuration file using the EXTRAPYPI_CONFIG env variable.

If the env variable is not set, default settings will be used.

This file is a pure python file, that's mean that you can also include python code in here, for example for packages location

Warning: For security reasons you should at least change the secret key

Note: If you use anything else than sqlite, you must install correct database drivers like psycopg2 or pymysql for example. Since we use SQLAlchemy you can use any compliant database, but we only test sqlite, mysql and postgresql

Note: You can also override all settings of flask extensions used by extra-pypi even if there are not here

For quickstart you can generate a sample configuration file using `start` command like this

```
extrapypi start --filename myconfig.cfg
```

Generated file will have the following content

```
# Database connexion string
SQLALCHEMY_DATABASE_URI = "sqlite:///extrapypi.db"

# Update this secret key for production !
SECRET_KEY = "changeit"

# Storage settings
```

```
# You need to update at least packages_root setting
STORAGE_PARAMS = {
    'packages_root': "/path/to/my/packages"
}
```

Configuration options

NAME	Description
BASE_DIR	Base directory, by default used by SQLALCHEMY_URI and PACKAGES_ROOT
SQLALCHEMY_URI	SQLAlchemy connexion string
DEBUG	Enable debug mode
STATIC_URL	Url for static files
SECRET_KEY	Secret key used for the application, you must update this
STORAGE	Storage class name to use
STORAGE_PARAMS	Storage class parameters, see specific storages documentation for more details
DASHBOARD	You can disable dashboard if you set it to FALSE
LOGGING_CONFIG	Logger configuration, using standard python dict config

Defaut logging config look like this

```
LOGGING_CONFIG = {
 'version': 1,
 'root': {
     'level': 'NOTSET',
     'handlers': ['default'],
 },
 'formatters': {
     'verbose': {
         'format': '[%(asctime)s: %(levelname)s / %(name)s] %(message)s',
     },
 },
 'handlers': {
     'default': {
         'level': 'INFO',
         'class': 'logging.StreamHandler',
         'formatter': 'verbose',
     },
 },
 'loggers': {
     'extrapypi': {
         'handlers': ['default'],
         'level': 'WARNING',
         'propagate': False,
     },
     'alembic.runtime.migration': {
         'handlers': ['default'],
         'level': 'INFO',
         'propagate': False
     },
 }
}
```

2.3 Deployment

You can run extrapypi with run command, but since it uses flask debug server, it's not suited for production.

But we provide a wsgi entry point to make it easier to run extrapypi using python wsgi server like gunicorn or uwsgi.

2.3.1 Gunicorn

Simple example using gunicorn

```
EXTRAPYPI_CONFIG=/path/to/myconfig.cfg gunicorn extrapypi.wsgi:app
```

Full example using systemd and nginx (based on http://docs.gunicorn.org/en/stable/deploy.html#systemd)

/etc/systemd/system/extrapypi.service

```
[Unit]
Description=extrapypi daemon
Requires=extrapypi.socket
After=network.target

[Service]
Environment=EXTRAPYPI_CONFIG=/path/to/myconfig.cfg
PIDFile=/run/gunicorn/pid
User=myuser
Group=myuser
RuntimeDirectory=gunicorn
ExecStart=/path/to/gunicorn --pid /run/gunicorn/pid   \
       --bind unix:/run/gunicorn/socket extrapypi.wsgi:app
       ExecReload=/bin/kill -s HUP $MAINPID
       ExecStop=/bin/kill -s TERM $MAINPID
       PrivateTmp=true
```

[Install] WantedBy=multi-user.target

/etc/systemd/system/extrapypi.socket

```
[Unit]
Description=extrapypi socket

[Socket]
ListenStream=/run/gunicorn/socket

[Install]
WantedBy=sockets.target
```

Next, enable and start the socket and service

```
systemctl enable extrapypi.socket
systemctl start extrapypi.service
```

Last step is to configure nginx as a reverse proxy, basic configuration will look like this

```
...
http {
  server {
      listen          8000;
```

```
      server_name     127.0.0.1;
      location / {
          proxy_pass http://unix:/run/gunicorn/socket;
      }
  }
}
...
```

2.3.2 Uwsgi

Simple example using uwsgi

```
EXTRAPYPI_CONFIG=/path/to/myconfig.cfg uwsgi --http 0.0.0.0:8000 --module extrapypi.
↪wsgi:app
```

Full example using systemd and nginx (based on http://uwsgi-docs.readthedocs.io/en/latest/Systemd.html)

/etc/systemd/system/extrapypi.socket

```
[Unit]
Description=Socket for extrapypi

[Socket]
ListenStream=/var/run/uwsgi/extrapypi.socket
SocketUser=myuser
SocketGroup=myuser
SocketMode=0660

[Install]
WantedBy=sockets.target
```

/etc/systemd/system/extrapypi.service

```
[Unit]
Description=%i uWSGI app
After=syslog.target

[Service]
ExecStart=/path/to/uwsgi \
            --socket /var/run/uwsgi/extrapypi.socket \
            --module extrapypi.wsgi:app
User=myuser
Group=myuser
Restart=on-failure
KillSignal=SIGQUIT
Type=notify
StandardError=syslog
NotifyAccess=all
```

Note: you can also add your own ini file for uwsgi configuration

Next, enable and start the socket and service

```
systemctl enable extrapypi.socket
systemctl start extrapypi.service
```

Last step is to configure nginx as a reverse proxy, basic configuration will look like this

```
...
http {
  server {
      listen          8000;
      server_name     127.0.0.1;
      location / {
        uwsgi_pass unix:///var/run/uwsgi/extrapypi.socket;
        include uwsgi_params;
     }
  }
}
...
```

2.3.3 Monitoring

To make it simpler for you to check if extrapypi server is running with your monitoring tools, we provide a simple endpoint `/ping` that will always return `pong` with status code `200`. You must call this endpoint with `GET` http verb

2.4 Configuring pip to work with extrapypi

2.4.1 Uploading packages

extrapypi is compliant with setuptools / twine, you just need to update your `.pypirc`

```
[distutils]
index-servers =
    local

[local]
username=myuser
password=mypassword
repository=https://myextrapypiurl/simple/
```

That's it, you can now upload packages to your extrapypi instance

Using setuptools

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

Or twine

```
twine upload -r local dist/extra_pypi-0.1-py3.5.egg
```

2.4.2 Installing packages

Two choices here :

Using CLI argument when calling pip

```
pip install extrapypi --extra-index-url https://user:password@myextrapypiurl/simple/
```

Or update your `pip.conf` file

```
[global]
extra-index-url = https://user:password@myextrapypiurl/simple/
```

2.5 API Documentation

2.5.1 extrapypi app

extrapypi.app module

Extrapypi app

Used to create application. This can be imported for wsgi file for uwsgi or gunicorn.

By default, application will look for a EXTRAPYPI_CONFIG env variable to load configuration file, but you can also pass a config parameter. The configuration files are loaded in the following order :

- Load default configuration
- If testing is set to True, load config_test.py and nothing else
- If config parameter is not None, use it and don't load env variable config file
- If config parameter is None, try to load env variable

You can create a wsgi file like this for running gunicorn or uwsgi :

```
from extrapypi.app import create_app

app = create_app()
```

Or add any extra code if needed

`extrapypi.app.`**`configure_app`**(*app*, *testing*, *config*)
Set configuration for application

Configuration will be loaded in the following order:

- test_config if testing is True
- else if config parameter is not None we load it
- else if env variable for config is set we use it

`extrapypi.app.`**`configure_extensions`**(*app*)
Init all extensions

For login manager, we also register callbacks here

`extrapypi.app.`**`configure_logging`**(*app*)
Configure loggers

`extrapypi.app.`**`create_app`**(*testing=False*, *config=None*)
Main application factory

`extrapypi.app.`**`register_blueprints`**(*app*)
Register all views for application

extrapypi.app.**register_filters**(*app*)
Register additionnal jinja2 filters

extrapypi.config module

Extrapypi configuration

All settings present here can be override with your own configuration file using the EXTRAPYPI_CONFIG env variable.

If the env variable is not set, default settings will be used.

This file is a pure python file, that's mean that you can also include python code in here, for example for packages location

Warning: For security reasons you should at least change the secret key

Note: If you use anything else than sqlite, you must install correct database drivers like psycopg2 or pymysql for example. Since we use SQLAlchemy you can use any compliant database, but we only test sqlite, mysql and postgresql

Note: You can also override all settings of flask extensions used by extra-pypi even if there are not here

For quickstart you can generate a sample configuration file using `start` command like this

```
extrapypi start --filename myconfig.cfg
```

Generated file will have the following content

```
# Database connexion string
SQLALCHEMY_DATABASE_URI = "sqlite:///extrapypi.db"

# Update this secret key for production !
SECRET_KEY = "changeit"

# Storage settings
# You need to update at least packages_root setting
STORAGE_PARAMS = {
    'packages_root': "/path/to/my/packages"
}
```

Configuration options

NAME	Description
BASE_DIR	Base directory, by default used by SQLALCHEMY_URI and PACKAGES_ROOT
SQLALCHEMY_URI	SQLAlchemy connexion string
DEBUG	Enable debug mode
STATIC_URL	Url for static files
SECRET_KEY	Secret key used for the application, you must update this
STORAGE	Storage class name to use
STORAGE_PARAMS	Storage class parameters, see specific storages documentation for more details
DASHBOARD	You can disable dashboard if you set it to FALSE
LOGGING_CONFIG	Logger configuration, using standard python dict config

Defaut logging config look like this

```
LOGGING_CONFIG = {
 'version': 1,
 'root': {
     'level': 'NOTSET',
     'handlers': ['default'],
 },
 'formatters': {
     'verbose': {
         'format': '[%(asctime)s: %(levelname)s / %(name)s] %(message)s',
     },
 },
 'handlers': {
     'default': {
         'level': 'INFO',
         'class': 'logging.StreamHandler',
         'formatter': 'verbose',
     },
 },
 'loggers': {
     'extrapypi': {
         'handlers': ['default'],
         'level': 'WARNING',
         'propagate': False,
     },
     'alembic.runtime.migration': {
         'handlers': ['default'],
         'level': 'INFO',
         'propagate': False
     },
 }
}
```

2.5.2 commons

filters module

Jinja2 additionnal filters

extrapypi.commons.filters.**tohtml**(*s*)

Convert rst string to raw html

Used for display long description of releases

login module

Module handling all Flask-Login logic and handlers

extrapypi.commons.login.**load_user_from_request**(*request*)
Used to identify a request from pip or twine when downloading / uploading packages and releases

extrapypi.commons.login.**on_identity_loaded**(*sender*, *identity*)
Load rights for flask-principal

Handle only role need and user need

extrapypi.commons.login.**unauthorized**()

extrapypi.commons.login.**user_loader**(*user_id*)
Default user handler, from Flask-Login documentation

packages module

Packages utils.

This module export packages logic outside of the views

extrapypi.commons.packages.**create_package**(*name*, *summary*, *store*)
Create a package for a given release if the package don't exists already

Note: Maintainer and installer cannot create packages

Parameters

- **data** (*dict*) – request data to use to create package
- **storage** (*extrapypi.storage.BaseStorage*) – storage object to use

Raises PermissionDenied

extrapypi.commons.packages.**create_release**(*data*, *config*, *files*)
Register and save a new release

Since pypi itself don't support pre-registration anymore, we don't

Note: Installers cannot create a new release

If a release with same version number and package exists, we return it :param dict data: request data for registering package :param dict config: current app config :raises: PermissionDenied

extrapypi.commons.packages.**create_release_from_source**(*metadata*, *user*)
Create a new release from a raw file. Used for import of existing packages into database

Warning: This function does not check any permissions since it's never called from web ui

If a release already exists, it does nothing

Parameters

- **metadata** (*dict*) – metadata of the package

- **user** (*extrapypi.models.User*) – user to use as maintainer

extrapypi.commons.packages.**get_store**(*name*, *params*)
Utility function to get correct storage class based on its name

Parameters

- **name** (*str*) – name of the storage
- **params** (*dict*) – storage params from application config

Returns Correct storage class instance, passing params to constructor

Return type *BaseStorage*

Raises AttributeError

permission module

Permissions and needs helpers

Roles logic is defined like this :

admin > developer > maintainer > installer

2.5.3 dashboard

views module

Views for dashboard

All dashboard blueprint can be disabled if you set `DASHBOARD = False` in configuration

extrapypi.dashboard.views.**create_user**()
Create a new user

extrapypi.dashboard.views.**delete_package**(*package_id*)
Delete a package, all its releases and all files and directory associated with it

extrapypi.dashboard.views.**delete_user**(*user_id*)
Delete a user and redirect to dashboard

extrapypi.dashboard.views.**index**()
Dashboard index, listing packages from database

extrapypi.dashboard.views.**login**()
Login view

Will redirect to dashboard index if login is successful

extrapypi.dashboard.views.**logout**()
Logout view

Will redirect to login view after logout current user

extrapypi.dashboard.views.**package**(*package*)
Package detail view

extrapypi.dashboard.views.**release**(*package*, *release_id*)
Specific release view

extrapypi.dashboard.views.**search**()
Search page

Will use SQL Like syntax to search packages

extrapypi.dashboard.views.**user_detail**(*user_id*)
View to update user from admin account

extrapypi.dashboard.views.**users_list**()
List user in dashboard

2.5.4 forms

user module

WTForms forms class declaration for users

class extrapypi.forms.user.**LoginForm**(*formdata=<object object>*, ***kwargs*)
Bases: flask_wtf.form.FlaskForm

password = <UnboundField(PasswordField, ('password',), {'validators': [<wtforms.valida

remember = <UnboundField(BooleanField, ('Remember me',), {})>

username = <UnboundField(StringField, ('username',), {'validators': [<wtforms.validat

class extrapypi.forms.user.**PasswordForm**(*formdata=<object object>*, ***kwargs*)
Bases: flask_wtf.form.FlaskForm

confirm = <UnboundField(PasswordField, ('Repeat password',), {})>

current = <UnboundField(PasswordField, ('Current password',), {'validators': [<wtforms

password = <UnboundField(PasswordField, ('New password',), {'validators': [<wtforms.va

class extrapypi.forms.user.**UserCreateForm**(*formdata=<object object>*, ***kwargs*)
Bases: flask_wtf.form.FlaskForm

confirm = <UnboundField(PasswordField, ('Repeat password',), {})>

email = <UnboundField(EmailField, ('email',), {'validators': [<wtforms.validators.Data

is_active = <UnboundField(BooleanField, ('active',), {})>

password = <UnboundField(PasswordField, ('password',), {'validators': [<wtforms.valida

role = <UnboundField(SelectField, ('role',), {'choices': []})>

username = <UnboundField(StringField, ('username',), {'validators': [<wtforms.validat

class extrapypi.forms.user.**UserForm**(*formdata=<object object>*, ***kwargs*)
Bases: flask_wtf.form.FlaskForm

email = <UnboundField(EmailField, ('email',), {'validators': [<wtforms.validators.Data

is_active = <UnboundField(BooleanField, ('active',), {})>

role = <UnboundField(SelectField, ('role',), {'choices': []})>

username = <UnboundField(StringField, ('username',), {'validators': [<wtforms.validat

2.5.5 models

package module

class extrapypi.models.package.**Package**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Model

Represent a simple package

created_at

id

latest_release

maintainers

name

sorted_releases

summary

updated_at

release module

class extrapypi.models.release.**Release**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Model

created_at

description

download_url

home_page

id

keywords

package

package_id

updated_at

version

types module

Custom SQLAlchemy types / variants

Use mysql.LONGTEXT instead of mysql.TEXT for UnicodeText type

user module

class extrapypi.models.user.**User**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Model

ROLES = ['admin', 'developer', 'installer', 'maintainer']

email

get_id()

id

is_active

is_admin

is_anonymous

is_authenticated

password_hash

role

username

validate_role(*key*, *role*)

2.5.6 simple

views module

Views for handling simple index like original pypi

extrapypi.simple.views.**download_package**(*package*, *source*)
Return a package file from storage

extrapypi.simple.views.**package_view**(*package*)
List all files avaible for a package

extrapypi.simple.views.**simple**()
Simple view index used to list or upload packages

Used to list packages. Simple index is generated on the fly based on SQL data

2.5.7 storage

base module

Base Storage

BaseStorage define all methods needed by storages. All storage must be inherited from `BaseStorage` and implement the following methods

- delete_package
- delete_release
- create_package
- create_release
- get_files
- get_file

Storages classes handle all packages and releases operation outside of the SQL database, this include storage of packages sources, listing of files, removing deleted packages, etc.

class extrapypi.storage.base.**BaseStorage**(***kwargs*)
Bases: object

Base class for storage drivers, should be inherited by all sub-classes

In the constructor, kwargs are used to pass settings to the driver. By default it will set an attribute for each item in kwargs

NAME = None

create_package(*package*)
Must create a new location for a package

Parameters **package** (*models.Package*) – new package that need an emplacement

Returns True if creation successful, else return False

Return type bool

create_release(*package*, *release_file*)
Must copy release_file to the correct location

Note: release_file will be a werkzeug.datastructures.FileStorage object

Parameters

- **package** (*models.Package*) – package for the release
- **release_file** (*FileStorage*) – release file to save

delete_package(*package*)
Must delete an entire package

Parameters **package** (*models.Package*) – package to delete

Returns True if deletion is successful or False

Return type bool

delete_release(*package*, *version*)
Must delete all files of a package version

Parameters

- **models.Package** – package to delete
- **version** (*str*) – version to delete

Returns True if deletion is successful or False

Return type bool

get_file(*package*, *file*, *release=None*)
Must return a given file

Returned value will be directly send to Flask.send_file in most cases, be sure that return format is compatible with this function

Parameters

- **package** (*models.Package*) – package objet
- **file** (*str*) – file name to find

get_files(*package*, *release=None*)
Must return all files for a given package / release

Parameters

- **package** (*models.Package*) – package object for which we want files
- **release** (*models.Release*) – for filter files returned based on release

Returns list of all avaible files for this package or None if an error append

Return type list

get_releases_metadata()
Must return an iterable of tuples containing name of the package and release metadata

Returns list of all distributions contained in storage

Return type iterable

local module

LocalStorage

Simple local storage that create directories for packages and put releases files in it.

class extrapypi.storage.local.**LocalStorage**(*packages_root=None*)
Bases: *extrapypi.storage.base.BaseStorage*

NAME = 'LocalStorage'

create_package(*package*)
Create new directory for a given package

create_release(*package*, *release_file*)
Copy release file inside package directory

If package directory does not exists, it will create it before

delete_package(*package*)
Delete entire package directory

delete_release(*package*, *version*)
Delete all files matching specified version

get_file(*package*, *file*, *release=None*)
Get a single file from filesystem

get_files(*package*, *release=None*)
Get all files associated to a package

If release is not None, it will filter files on release version, based on a regex

get_releases_metadata()
List all releases metadata from PACKAGES_ROOT

Returns generator

Return type list

2.5.8 user package

views module

Views for self-user management

If you set `DASHBOARD = False` in settings, this blueprint will also be disabled

`extrapypi.user.views.`**`update_password`**`()`
Update current logged user password

`extrapypi.user.views.`**`update_user`**`()`
Update current logged user

2.5.9 utils

views module

Utils views

`extrapypi.utils.views.`**`ping`**`()`
Simple view used to monitor extrapypi server

Python Module Index

e

Index

H

I

K

L

M

N

O

P

R

S

T

U

V