pypicloud

Release 0.5.3

May 01, 2017

Contents

This is an implementation of the PyPI server for hosting your own python packages. It stores the packages in S3 and dynamically generates links to them for pip.

After generating the S3 urls, pypicloud caches them in a database. Subsequent requests to download packages will use the already-generated urls in the db. Pypicloud supports using SQLAlchemy, Redis, or DynamoDB as the cache.

Pypicloud was designed to be fast and easy to replace in the case of server failure. Simply copy your config.ini file to a new server and run pypicloud there. The only data that needs to be persisted is in S3, which handles the redundancy requirements for you.

Code lives here: https://github.com/stevearc/pypicloud

CHAPTER 1

User Guide

Getting Started

Installation

First create and activate a virtualenv to contain the installation:

```
$ virtualenv mypypi
New python executable in mypypi/bin/python
Installing setuptools.............done.
Installing pip...............done.
$ source mypypi/bin/activate
(mypypi)$
```

Now install pypicloud and waitress. To get started, we're using waitress as the WSGI server because it's easy to set up.

```
(mypypi)$ pip install pypicloud[server]
```

AWS

If you have not already, create an access key and secret by following the AWS guide

The default configuration should work, but if you get permission errors or 403's, you will need to set a policy on your bucket.

Configuration

Generate a default server configuration

```
(mypypi)$ ppc-make-config -t server.ini
```

Warning: Note that this configuration should only be used for testing. If you want to set up your server for production, read the section on *deploying*.

Running

Now you can run the server using pserve

```
(mypypi)$ pserve server.ini
```

The server is now running on port 6543. You can view the web interface at http://localhost:6543/

Installing Packages

After you have the webserver started, you can install packages using:

```
pip install -i http://localhost:6543/simple/ PACKAGE1 [PACKAGE2 ...]
```

If you want to configure pip to always use PyPI Cloud, you can put your preferences into the `$HOME/.pip/pip.conf` file:

```
[global]
index-url = http://localhost:6543/simple/
```

Uploading Packages

To upload packages, you will need to add your server as an index server inside your `$HOME/.pypirc`:

```
[distutils]
index-servers = pypicloud

[pypicloud]
repository: http://localhost:6543/simple/
username: <<username>>
password: <<password>>
```

Now to upload a package you should run:

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

Searching Packages

After packages have been uploaded, you can search for them via pip:

```
pip search -i http://localhost:6543/simple/ QUERY1 [QUERY2 ...]
```

If you want to configure pip to use PyPI Cloud for search, you can update your preferences in the `$HOME/.pip/pip.conf` file:

```
[search]
index = http://localhost:6543/simple/
```

Note that this will ONLY return results from the PyPi Cloud repository. The official PyPi repository will not be queried.

Configuration Options

This is a list of all configuration parameters for pypicloud

PyPICloud

pypi.fallback

Argument: {'redirect', 'cache', 'none'}, optional

This option defines what the behavior is when a requested package is not found in the database. (default 'redirect')

`redirect` - Return a 302 to the package at the `fallback_url`.

`cache` - Download the package from `fallback_url`, store it in the backend, and serve it. User must have `cache_update` permissions.

`none` - Return a 404

See also *pypi.always_show_upstream* below.

See fallback_detail for more detail on exactly how each fallback option will function.

pypi.always_show_upstream

Argument: bool, optional

Default `False`.

This adjusts the fallback behavior when one or more versions of the requested package are stored in pypicloud. If `False`, pypicloud will only show the client the versions that are stored. If `True`, the local versions will be shown with the versions found at the `fallback_url`.

pypi.fallback_url

Argument: string, optional

The index server to handle the behavior defined in `pypi.fallback` (default https://pypi.python.org/simple)

pypi.default_read

Argument: list, optional

List of groups that are allowed to read packages that have no explicit user or group permissions (default ['authenticated'])

pypi.default_write

Argument: list, optional

List of groups that are allowed to write packages that have no explicit user or group permissions (default no groups, only admin users)

pypi.cache_update

Argument: list, optional

Only used when `pypi.fallback = cache`. This is the list of groups that are allowed to trigger the operation that fetches packages from `fallback_url`. (default ['authenticated'])

pypi.allow_overwrite

Argument: bool, optional

Allow users to upload packages that will overwrite an existing version (default False)

pypi.realm

Argument: string, optional

The HTTP Basic Auth realm (default 'pypi')

pypi.use_fallback (deprecated)

Argument: bool, optional

Replaced by `pypi.fallback`. Setting to `True` has no effect. Setting to `False` will set `pypi.fallback = none`.

Storage

pypi.storage

Argument: string, optional

A dotted path to a subclass of *IStorage*. The default is *FileStorage*. Each storage option may have additional configuration options. Documentation for the built-in storage backends can be found at *Storage Backends*.

Cache

pypi.db

Argument: string, optional

A dotted path to a subclass of *ICache*. The default is *SQLCache*. Each cache option may have additional configuration options. Documentation for the built-in cache backends can be found at *Caching Backends*.

Access Control

pypi.auth

Argument: string, optional

A dotted path to a subclass of *IAccessBackend*. The default is *ConfigAccessBackend*. Each backend option may have additional configuration options. Documentation for the built-in backends can be found at *Access Control*.

Beaker

Beaker is the session manager that handles user auth for the web interface. There are many configuration options, but these are the only ones you need to know about.

session.encrypt_key

Argument: string

Encryption key to use for the AES cipher. Here is a reasonable way to generate one:

```
$ python -c 'import os, base64; print base64.b64encode(os.urandom(32))'
```

session.validate_key

Argument: string

Validation key used to sign the AES encrypted data.

session.secure

Argument: bool, optional

If True, only set the session cookie for HTTPS connections (default False). When running a production server, make sure this is always set to `true`.

Storage Backends

The storage backend is where the actual package files are kept.

Files

This will store your packages in a directory on disk. It's much simpler and faster to set up if you don't need the reliability and scalability of S3.

Set `pypi.storage = file` OR `pypi.storage = pypicloud.storage.FileStorage` OR leave it out completely since this is the default.

storage.dir

Argument: string

The directory where the package files should be stored.

S3

This option will store your packages in S3.

Set `pypi.storage = s3` OR `pypi.s3 = pypicloud.storage.S3Storage`

storage.bucket

Argument: string

The name of the S3 bucket to store packages in.

> **Warning:** Your bucket must not have "." in it. Amazon's SSL certificate for S3 urls is only valid for *.s3.amazonaws.com

storage.access_key, storage.secret_key

Argument: string, optional

Your AWS access key id and secret access key. If they are not specified then pypicloud will attempt to get the values from the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.

storage.region

Argument: string, semi-optional

This is required if your bucket is in a new region, such as `eu-central-1`. If your bucket does not yet exist, it will be created in this region on startup. If blank, the classic US region will be used.

storage.host

Argument: string, optional

Specify S3 compatible API fqdn/IP. On other systems, this variable could be called an endpoint.

storage.is_secure

Argument: boolean, optional

Use secure connection (https) or not (http). Default is True.

storage.calling_format

Argument: string, optional

Choose how to call the S3 API. Supported formats are SubdomainCallingFormat, VHostCallingFormat, OrdinaryCallingFormat and ProtocolIndependentOrdinaryCallingFormat. For example, not all S3 compatible API use <bucket>.<host> format, so put this value to OrdinaryCallingFormat.

storage.prefix

Argument: string, optional

If present, all packages will be prefixed with this value when stored in S3. Use this to store your packages in a subdirectory, such as "packages/"

storage.prepend_hash

Argument: bool, optional

Prepend a 4-letter hash to all S3 keys (default True). This helps S3 load balance when traffic scales. See the AWS documentation on the subject.

storage.expire_after

Argument: int, optional

How long (in seconds) the generated S3 urls are valid for (default 86400 (1 day)). In practice, there is no real reason why these generated urls need to expire at all. S3 does it for security, but expiring links isn't part of the python package security model. So in theory you can bump this number up.

storage.redirect_urls

Argument: bool, optional

The short story: set this to `true` if you only use pip and don't have to support easy_install. It will dramatically speed up your server.

The long story: redirect_detail

storage.server_side_encryption

Argument: bool, optional

Enables AES-256 transparent server side encryption. See the AWS documention. Default is False.

CloudFront

This option will store your packages in S3 but use CloudFront to deliver the packages. This is an extension of the S3 storage backend and require the same settings as above, but also the settings listed below.

Set `pypi.storage = cloudfront` OR `pypi.s3 = pypicloud.storage.CloudFrontS3Storage`

storage.cloud_front_domain

Argument: string

The CloudFront domain you have set up. This CloudFront distribution must be set up to use your S3 bucket as the origin.

Example: `https://dabcdefgh12345.cloudfront.net`

storage.cloud_front_key_id

Argument: string, optional

If you want to protect your packages from public access you need to set up the CloudFront distribution to use signed URLs. This setting specifies the key id of the CloudFront key pair that is currently active on your AWS account.

storage.cloud_front_key_file

Argument: string, optional

Only needed when setting up CloudFront with signed URLs. This setting should be set to the full path of the CloudFront private key file.

storage.cloud_front_key_string

Argument: string, optional

The same as `cloud_front_key_file`, but contains the raw private key instead of a path to a file.

Caching Backends

PyPICloud stores the packages in a storage backend (typically S3), but that backend is not necessarily efficient for frequently reading metadata. So instead of hitting S3 every time we need to find a list of package versions, we store all that metadata in a cache. The cache does not have to be backed up because it is only a local cache of data that is permanently stored in the storage backend.

SQLAlchemy

Set `pypi.db = sql` OR `pypi.db = pypicloud.cache.SQLCache` OR leave it out completely since this is the default.

db.url

Argument: string

The database url to use for the caching database. Should be a SQLAlchemy url

- sqlite: `sqlite:///%(here)s/db.sqlite`
- sqlite (in-memory): `sqlite://`
- mysql: `mysql://root@127.0.0.1:3306/pypi?charset=utf8mb4`
- postgres: `postgresql://postgres@127.0.0.1:5432/postgres`

Warning: You must specify the `charset=` parameter if you're using MySQL, otherwise it will choke on unicode package names. If you're using 5.5.3 or greater you can specify the `utf8mb4` charset, otherwise use `utf8`.

Redis

Set `pypi.db = redis` OR `pypi.db = pypicloud.cache.RedisCache`

You will need to `pip install redis` before running the server.

db.url

Argument: string

The database url to use for the caching database. The format looks like this: `redis://username:password@localhost:6379/0`

DynamoDB

Set `pypi.db = dynamo` OR `pypi.db = pypicloud.cache.dynamo.DynamoCache`

You will need to `pip install pypicloud[dynamo]` before running the server.

db.region

Argument: string

The AWS region to use for the cache tables. You must specify either this or `db.host`.

db.access_key, db.secret_key

Argument: string, optional

Your AWS access key id and secret access key. If they are not specified then pypicloud will attempt to get the values from the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.

db.namespace

Argument: string, optional

If specified, all of the created Dynamo tables will have this as a prefix in their name. Useful to avoid name collisions.

db.host

Argument: string

The hostname to connect to. This is normally used to connect to a DynamoDB Local instance. You must specify either this or `db.region`.

db.port

Argument: int, optional

The port to connect to when using `db.host` (default 8000)

db.secure

Argument: bool, optional

Force https connection when using `db.host` (default False)

Access Control

PyPICloud has a complete access control system that allows you to fine-tune who has access to your packages. There are several choices for where to store your user credentials and access rules.

Users and Groups

The access control uses a combination of users and groups. A group is a list of users. There are also *admin* users, who always have read/write permissions for everything, and can do a few special operations besides. There are two special groups:

- `everyone` - This group refers to any anonymous user making a request
- `authenticated` - This group refers to all logged-in users

You will never need to specify the members of these groups, as membership is automatic.

Config File

The simplest access control available (which is the default) pulls user, group, and package permission information directly from the config file.

Here is a sample configuration to get you started:

```
# USERS
# user: stevearc, pass: gunface
user.stevearc = $5$rounds=80000$yiWi67QBJLDTvbI/$d6qIG/bIoM3hp0lxH8v/
↪vzxg8Qc4CJbxbxiUH4MlnE7
# user: dsa, pass: paranoia
user.dsa = $5$rounds=80000$U/lot7eW6gFvuvna$KDyrQvi40XXWzMRkBq1Z/0odJEXzqUVNaPIArL/
↪W0s6
# user: donlan, pass: osptony
user.donlan = $5$rounds=80000$Qjz9eRNXrybydMz.$PoD.5vAR9Z2IYlOCPYbza1cKvQ.
↪8kuz1cP0zKl314g0

# GROUPS
group.sharkfest =
    stevearc
    dsa
group.brotatos =
    donlan
    dsa
```

```
# PACKAGES
package.django_unchained.user.stevearc = rw
package.django_unchained.group.sharkfest = rw

package.polite_requests.user.dsa = rw
package.polite_requests.group.authenticated = r
package.polite_requests.group.brotatos = rw

package.pyramid_head.group.brotatos = rw
package.pyramid_head.group.everyone = r
```

Here is a table that describes who has what permissions on these packages. Note that if the entry is `none`, that user will not even see the package listed, depending on your `pypi.default_read` and `pypi.default_write` settings.

User	django_unchained	polite_requests	pyramid_head
stevearc	rw (user)	r (authenticated)	r (everyone)
dsa	rw (sharkfest)	rw (user)	rw (brotatos)
donlan	none	rw (brotatos)	rw (brotatos)
everyone	none	none	r (everyone)

Configuration

Set `pypi.auth = config` OR `pypi.auth = pypicloud.access.ConfigAccessBackend` OR leave it out completely since this is the default.

`user.<username>`

Argument: string

Defines a single user login. You may specify any number of users in the file. Use `ppc-gen-password` to create the password hashes.

`package.<package>.user.<user>`

Argument: {`r`, `rw`}

Give read or read/write access on a package to a single user.

`package.<package>.group.<group>`

Argument: {`r`, `rw`}

Give read or read/write access on a package to a group of users. The group must be defined in a `group.<group>` field.

`auth.admins`

Argument: list

Whitespace-delimited list of users with admin privileges. Admins have read/write access to all packages, and can perform maintenance tasks.

group.<group>

Argument: list

Whitespace-delimited list of users that belong to this group. Groups can have separately-defined read/write permissions on packages.

auth.zero_security_mode (deprecated)

Argument: bool, optional

Replaced by `pypi.default_read` and `pypi.default_write`. If enabled, will set `pypi.default_read = everyone` and `pypi.default_write = authenticated`.

SQL Database

You can opt to store all user and group permissions inside a SQL database. The advantages are that you can dynamically change these permissions using the web interface. The disadvantages are that this information is not stored anywhere else, so unlike the *cache database*, it actually needs to be backed up. There is an import/export command *that makes this easy*.

After you set up a new server using this backend, you will need to use the web interface to create the initial admin user.

Configuration

Set `pypi.auth = sql` OR `pypi.auth = pypicloud.access.sql.SQLAccessBackend`

auth.db.url

Argument: string

The database url to use for storing user and group permissions. This may be the same database as `db.url` (if you are also using the SQL caching database).

Remote Server

This implementation allows you to delegate all access control to another server. If you already have an application with a user database, this allows you to use that data directly.

You will need to `pip install requests` before running the server.

Configuration

Set `pypi.auth = remote` OR `pypi.auth = pypicloud.access.RemoteAccessBackend`

auth.backend_server

Argument: string

The base host url to connect to when fetching access data (e.g. http://myserver.com)

auth.user

Argument: string, optional

If provided, the requests will use HTTP basic auth with this user

auth.password

Argument: string, optional

If `auth.user` is provided, this will be the HTTP basic auth password

auth.uri.verify

Argument: string, optional

The uri to hit when verifying a user's password (default `/verify`).

params: `username`, `password`

returns: `bool`

auth.uri.groups

Argument: string, optional

The uri to hit to retrieve the groups a user is a member of (default `/groups`).

params: `username`

returns: `list`

auth.uri.group_members

Argument: string, optional

The uri to hit to retrieve the list of users in a group (default `/group_members`).

params: `group`

returns: `list`

auth.uri.admin

Argument: string, optional

The uri to hit to determine if a user is an admin (default `/admin`).

params: `username`

returns: `bool`

auth.uri.group_permissions

Argument: string, optional

The uri that returns a mapping of groups to lists of permissions (default `/group_permissions`). The permission lists can contain zero or more of ('read', 'write').

params: `package`

returns: `dict`

auth.uri.user_permissions

Argument: string, optional

The uri that returns a mapping of users to lists of permissions (default `/user_permissions`). The permission lists can contain zero or more of ('read', 'write').

params: `package`

returns: `dict`

auth.uri.user_package_permissions

Argument: string, optional

The uri that returns a list of all packages a user has permissions on (default `/user_package_permissions`). Each element is a dict that contains 'package' (str) and 'permissions' (list).

params: `username`

returns: `list`

auth.uri.group_package_permissions

Argument: string, optional

The uri that returns a list of all packages a group has permissions on (default `/group_package_permissions`). Each element is a dict that contains 'package' (str) and 'permissions' (list).

params: `group`

returns: `list`

auth.uri.user_data

Argument: string, optional

The uri that returns a list of users (default `/user_data`). Each user is a dict that contains a `username` (str) and `admin` (bool). If a username is passed to the endpoint, return just a single user dict that also contains `groups` (list).

params: `username`

returns: `list`

LDAP Authentication

You can opt to authenticate all users through a remote LDAP or compatible server. There is aggressive caching in the LDAP backend in order to keep chatter with your LDAP server at a minimum. If you experience a change in your LDAP layout, group modifications etc, restart your pypicloud process.

Note that you will need to `pip install pypicloud[ldap]` OR `pip install -e .[ldap]` (from source) in order to get the dependencies for the LDAP authentication backend.

At the moment there is no way for pypicloud to discern groups from LDAP, so it only has the built-in `admin`, `authenticated`, and `everyone` as the available groups. All authorization is configured using `pypi.default_read`, `pypi.default_write`, and `pypi.cache_update`.

Configuration

Set `pypi.auth = ldap` OR `pypi.auth = pypicloud.access.ldap_.LDAPAccessBackend`

auth.ldap.url

Argument: string

The LDAP url to use for remote verification. It should include the protocol and port, as an example: `ldap://10.0.0.1:389`

auth.ldap.service_dn

Argument: string

The FQDN of the LDAP service account used. A service account is required to perform the initial bind with. It only requires read access to your LDAP.

auth.ldap.service_password

Argument: string

The password for the LDAP service account.

auth.ldap.base_dn

Argument: string

The base DN under which all of your user accounts are organized in LDAP. Used in combination with the `all_user_search` to find all potential users.

auth.ldap.all_user_search

Argument: string

An LDAP search phrase, which when used with the `base_dn` results in a list of all users. As an example, this could be something like "(accountType=human)" depending on your organization's LDAP configuration.

auth.ldap.id_field

Argument: string

The field in the LDAP return when using `all_user_search` on `base_dn` to determine the account name from the record. As an example this could be "name", but it will depend on how your LDAP is set up.

auth.ldap.admin_field

Argument: string

The field to use in combination with `admin_dns` to determine the admin DNs from the search result. As an example, this could be "groupMembers", but again, will depend on your LDAP setup.

auth.ldap.admin_dns

Argument: list

A list of DNs to search for who should be considered an admin. It uses the `admin_field` to determine the source of admin DNs in the returned record(s).

auth.ldap.service_account

Argument: string, optional

A username for the service DN to self-authenticate as admin.

Deploying

This section is geared towards helping you deploy this server properly for production use.

@powellc has put together an Ansible playbook for pypicloud, which can be found here: https://github.com/powellc/ansible-pypicloud

Configuration

Remember when you generated a config file in *getting started*? Well we can do the same thing with a different flag to generate a default production config file.

```
$ ppc-make-config -p prod.ini
```

You should make extra certain that `session.secure` is `true` if the server is visible to the outside world. This goes hand-in-hand with ensuring that your server always uses HTTPS. If you set up access control and don't use HTTPS, someone will just sniff your credentials. And then you'll feel silly.

WSGI Server

You probably don't want to use waitress for your production server, though it will work fine for small deploys. I recommend using uWSGI with NGINX. It's fast and mature.

After creating your production config file, it will have a section for uwsgi. You can run uwsgi with

```
$ pip install uwsgi
$ uwsgi --ini-paste-logged prod.ini
```

Now uwsgi is running, but it's not speaking HTTP. It speaks a special uwsgi protocol that is used to communicate with nginx. Below is a sample nginx configuration that will listen on port 80 and send traffic to uwsgi.

```
server {
    listen 80;
    server_name pypi.myserver.com;
    location / {
        include uwsgi_params;
        uwsgi_pass 127.0.0.1:3031;
    }
}
```

Note: When using access control in production, you may need pip to pass up a username/password. Do that by just putting it into the url in the canonical way: `pip install mypkg -i https://username:password@pypi.myserver.com/simple/`

Warning: If you are using `pypi.fallback = cache`, make sure your uWSGI settings includes `enable-threads = true`. The package downloader uses threads.

Upgrading

New versions of PyPICloud may require action in up to two locations:

1. The cache database
2. The access control backend

Cache Database

This storage system is designed to be ephemeral. After an upgrade, all you need to do is rebuild the cache from the storage backend and that will apply any schema changes needed. You can use the "rebuild" button in the admin interface, or you can hit the *REST endpoint*.

Access Control

If something has changed in the formatting of the access control between versions, there should be a note inside the changelog. If so, you will need to export your current data and import it to the new version.

```
$ ppc-export config.ini -o acl.json.gz
$ pip install --upgrade pypicloud
$ # Make any necessary changes to the config.ini file
$ ppc-import config.ini -i acl.json.gz
```

Note that this system also allows you to migrate your access rules from one backend to another.

```
$ ppc-export old_config.ini | ppc-import new_config.ini
```

Migrating Packages

If you would like to change your storage backend, you will need to migrate your existing packages to the new location. Create a config file that uses the new storage backend, and then run:

```
ppc-migrate old_config.ini new_config.ini
```

This will find all packages stored in the old storage backend and upload them to the new storage backend.

Extending PyPICloud

Certain parts of PyPICloud were created to be pluggable. The storage backend, cache database, and access control backend can all be replaced very easily.

The steps for extending are:

1. Create a new implementation that subclasses the base class (*ICache*, *IStorage*, *IAccessBackend*/*IMutableAccessBackend*)
2. Put that implementation in a package and install that package in the same virtualenv as PyPICloud
3. Pass in a dotted path to that implementation for the approprate config field (e.g. `pypi.db`)

HTTP API

For all endpoints you may provide HTTP Basic Auth credentials. Here is a quick example that flushes and rebuilds the cache database:

```
curl https://myadmin:myadminpass@pypi.myserver.com/admin/rebuild
```

/simple/ (or /pypi/)

These endpoints are usually only used by `pip`

GET /simple/

Returns a webpage with links to all the pages for each unique package

Example:

```
curl myserver.com/simple/
```

POST /simple/

Upload a package

Parameters:

- `:action` - The only valid value is `'file_upload'`
- `name` - The name of the package being uploaded
- `version` - The version of the package being uploaded
- `content` (file) - The file object that contains the package data

Example:

```
curl -F ':action=file_upload' -F 'name=flywheel' -F 'version=0.1.0' \
-F 'content=@path/to/flywheel-0.1.0.tar.gz' myserver.com/simple/
```

GET /simple/<package>/

Returns a webpage with all links to all versions of this package.

If *fallback* is configured and the server does not contain the package, this will return either a `302` that points towards the fallback server (`redirect`), or a package index pulled from the fallback server (`cache`).

Example:

```
curl myserver.com/simple/flywheel/
```

/api/

These endpoints are used by the web interface

GET /api/package/[?verbose=true/false]

If `verbose` is False, return a list of all unique package names. If `verbose` is True, return a list of summarized data for each unique package name.

Parameters:

- `verbose` (bool) - Determines the return format (default False)

Example:

```
curl myserver.com/api/package/
curl myserver.com/api/package/?verbose=true
```

Sample Response

for `verbose=false`:

```
{
    "packages": [
        "flywheel",
        "pypicloud",
        "pyramid"
```

```
    ]
}
```

for `verbose=true`:

```
{
    "packages": [
        {
            "name": "flywheel",
            "stable": "0.1.0",
            "unstable": "0.1.0-2-g185e630",
            "last_modified": 1389945600
        },
        {
            "name": "pypicloud",
            "stable": "0.1.0",
            "unstable": "0.1.0-21-g4a739b0",
            "last_modified": 1390207478
        }
    ]
}
```

GET /api/package/<package>/

Get all versions of a package. Also returns if the user has write permissions for that package.

Example:

```
curl myserver.com/api/package/flywheel
```

Sample Response:

```
{
    "packages": [
        {
            "name": "flywheel",
            "last_modified": 1389945600
            "version": "0.1.0"
            "url": "https://pypi.s3.amazonaws.com/34c2/flywheel-0.1.0.tar.gz?
↪Signature=%2FSJidAjDkXbDojzXy8P1rFwe1kw%3D&Expires=1390262542"
        },
        {
            "name": "flywheel",
            "last_modified": 1390207478
            "version": "0.1.0-21-g4a739b0",
            "url": "https://pypi.s3.amazonaws.com/81f2/flywheel-0.1.0-21-g4a739b0.tar.
↪gz?Signature=%2FSJidAjDkXbDojzXy8P1rFwe1kw%3D&Expires=1390262542"
        },
    ],
    "write": true
}
```

POST /api/package/<package>/<filename>

Upload a package to the server. This is just a cleaner endpoint that does the same thing as the `POST /simple/` endpoint.

Parameters:

- `content` (file) - The file object that contains the package data

Example:

```
curl -F 'content=@path/to/flywheel-0.1.0.tar.gz' myserver.com/api/package/flywheel/
↪flywheel-0.1.0.tar.gz
```

DELETE /api/package/<package>/<filename>

Delete a package version from the server

Example:

```
curl -X DELETE myserver.com/api/package/flywheel/flywheel-0.1.0.tar.gz
```

POST /api/fetch

Fetch packages from the `fallback_url` and cache them. This is only used if `pypi.fallback = cache`.

Parameters:

- `requirements` (str) - Packages to update, in requirements.txt format (yes, with newlines)
- `wheel` (bool) - Fetch the wheel version of packages, if available (default `True`)
- `prerelease` (bool) - Fetch unstable versions if available (ex. '1.4a1') (default `False`)

Example:

```
curl -d 'requirements=requests>=2.2.0&wheel=true&prerelease=false' myserver.com/api/
↪fetch
```

PUT /api/user/<username>/

Register a new user account (if user registration is enabled). After registration the user will have to be confirmed by an admin.

If the server doesn't have any admins then the first user registered becomes the admin.

Parameters:

- `password` - The password for the new user account

Example:

```
curl -X PUT -d 'password=foobar' myserver.com/api/user/LordFoobar
```

POST /api/user/password

Change your password

Parameters:

- `old_password` - Your current password
- `new_password` - The password you are changing to

Example:

```
curl -d 'old_password=foobar&new_password=F0084RR' myserver.com/api/user/password
```

/admin/

These endpoints are used by the admin web interface. Most of them require you to be using a mutable *access backend*.

GET /admin/rebuild/

Flush the cache database and rebuild it by enumerating the storage backend

Example:

```
curl myserver.com/admin/rebuild/
```

GET /admin/acl.json.gz

Download the ACL as a gzipped-json file. This is equivalent to running `ppc-export`.

Example:

```
curl -o acl.json.gz myserver.com/admin/acl.json.gz
```

POST /admin/register/

Set whether registration is enabled or not

Parameters:

- `allow` (bool) - If True, allow new users to register

Example:

```
curl -d 'allow=true' myserver.com/admin/register/
```

GET /admin/pending_users/

Get a list of all users that are registered and need confirmation from an admin

Example:

```
curl myserver.com/admin/pending_users/
```

Sample Response:

```
[
    "LordFoobar",
    "TotallyNotAHacker",
    "Wat"
]
```

GET /admin/user/

Get a list of all users and their admin status

Example:

```
curl myserver.com/admin/user/
```

Sample Response:

```
[
    {
        "username": "LordFoobar",
        "admin": true
    },
    {
        "username": "stevearc",
        "admin": false
    }
]
```

GET /admin/user/<username>/

Get detailed data about a single user

Example:

```
curl myserver.com/admin/user/LordFoobar/
```

Sample Response:

```
{
    "username": "LordFoobar",
    "admin": true,
    "groups": [
        "cool_people",
        "group2"
    ]
}
```

GET /admin/user/<username>/permissions/

Get a list of packages that a user has explicit permissions on

Example:

```
curl myserver.com/admin/user/LordFoobar/permissions/
```

Sample Response:

```
[
    {
        "package": "flywheel",
        "permissions": ["read", "write"]
    },
    {
```

```
        "package": "pypicloud",
        "permissions": ["read"]
    }
]
```

DELETE /admin/user/<username>/

Delete a user

Example:

```
curl -X DELETE myserver.com/admin/user/chump/
```

POST /admin/user/<username>/approve/

Mark a pending user as approved

Example:

```
curl -X POST myserver.com/admin/user/LordFoobar/approve/
```

POST /admin/user/<username>/admin/

Grant or revoke admin privileges for a user.

Parameters:

- `admin` (bool) - If True, promote to admin. If False, demote to regular user.

Example:

```
curl -d 'admin=true' myserver.com/admin/user/LordFoobar/admin/
```

PUT /admin/user/<username>/group/<group>/

Add a user to a group

Example:

```
curl -X PUT myserver.com/admin/user/LordFoobar/group/cool_people/
```

DELETE /admin/user/<username>/group/<group>/

Remove a user from a group

Example:

```
curl -X DELETE myserver.com/admin/user/LordFoobar/group/cool_people/
```

GET /admin/group/

Get a list of all groups

Example:

```
curl myserver.com/admin/group/
```

Sample Response:

```
[
    "cool_people",
    "uncool_people",
    "marginally_cool_people"
]
```

GET /admin/group/<group>/

Get detailed information about a group

Example:

```
curl myserver.com/admin/group/cool_people
```

Sample Response:

```
{
    "members": [
        "LordFoobar",
        "stevearc"
    ],
    "packages": [
        {
            "package": "flywheel",
            "permissions": ["read", "write"]
        },
        {
            "package": "pypicloud",
            "permissions": ["read"]
        }
    ]
}
```

PUT /admin/group/<group>/

Create a new group

Example:

```
curl -X PUT myserver.com/admin/group/cool_people/
```

DELETE /admin/group/<group>/

Delete a group

Example:

```
curl -X DELETE myserver.com/admin/group/uncool_people/
```

GET /admin/package/<package>/

Get the user and group permissions for a package

Example:

```
curl myserver.com/admin/package/flywheel/
```

Sample Response:

```
{
    "user": [
        {
            "username": "LordFoobar",
            "permissions": ["read", "write"]
        },
        {
            "username": "stevearc",
            "permissions": ["read"]
        }
    ],
    "group": [
        {
            "group": "marginally_cool_people",
            "permissions": ["read"]
        },
        {
            "group": "cool_people",
            "permissions": ["read", "write"]
        }
    ]
}
```

PUT /admin/package/<package>/(user|group)/<name>/(read|write)/

Grant a permission to a user or a group on a package

Example:

```
curl -X PUT myserver.com/admin/package/flywheel/user/LordFoobar/read
curl -X PUT myserver.com/admin/package/flywheel/group/cool_people/write
```

DELETE /admin/package/<package>/(user|group)/<name>/(read|write)/

Revoke a permission for a user or a group on a package

Example:

```
curl -X DELETE myserver.com/admin/package/flywheel/user/LordFoobar/read
curl -X DELETE myserver.com/admin/package/flywheel/group/cool_people/write
```

Developing

The fast way to get set up:

```
wget https://raw.github.com/stevearc/devbox/0.1.0/devbox/unbox.py && \
python unbox.py git@github.com:stevearc/pypicloud
```

The slow way to get set up:

```
$ git clone git@github.com:stevearc/pypicloud
$ cd pypicloud
$ virtualenv pypicloud_env
$ . pypicloud_env/bin/activate
$ pip install -r requirements_dev.txt
$ pip install -e .
$ rm -r .git/hooks
$ ln -s ../git_hooks .git/hooks # This will run pylint before you commit
```

Run `ppc-make-config -d development.ini` to create a developer config file.

Now you can run the server with

```
$ pserve --reload development.ini
```

The unit tests require a redis server to be running on port 6379. Run unit tests with:

```
$ python setup.py nosetests
```

or:

```
$ tox
```

Changelog

If you are upgrading an existing installation, read *the instructions*

0.5.3 - 2017/4/30

- Bug fix: S3 uploads failing from web interface and when fallback=cache (issue 120)

0.5.2 - 2017/4/22

- Bug fix: The `/pypi` path was broken for viewing & uploading packages (issue 119)
- Update docs to recommend `/simple` as the install/upload URL
- Beaker session sets `invalidate_corrupt = true` by default

0.5.1 - 2017/4/17

- Bug fix: Deleting packages while using the Dynamo cache would sometimes remove the wrong package from Dynamo (issue 118)

0.5.0 - 2017/3/29

Upgrade breaks: SQL caching database. You will need to rebuild it.

- Feature: Pip search works now (pull 107)

0.4.6 - 2017/4/17

- Bug fix: Deleting packages while using the Dynamo cache would sometimes remove the wrong package from Dynamo (issue 118)

0.4.5 - 2017/3/25

- Bug fix: Access backend now works with MySQL family (pull 106)
- Bug fix: Return http 409 for duplicate upload to work better with twine (issue 112)
- Bug fix: Show upload button in interface if `default_write = everyone`
- Confirm prompt before deleting a user or group in the admin interface
- Do some basica sanity checking of username/password inputs

0.4.4 - 2016/10/5

- Feature: Add optional AWS S3 Server Side Encryption option (pull 99)

0.4.3 - 2016/8/2

- Bug fix: Rebuilding cache always ends up with correct name/version (pull 93)
- Feature: /health endpoint (nothing fancy, just returns 200) (issue 95)

0.4.2 - 2016/6/16

- Bug fix: Show platform-specific versions of wheels (issue 91)

0.4.1 - 2016/6/8

- Bug fix: LDAP auth disallows empty passwords for anonymous binding (pull 92)
- Config generator sets `pypi.default_read = authenticated` for prod mode

0.4.0 - 2016/5/16

Backwards incompatibility: This version was released to handle a change in the way pip 8.1.2 handles package names. If you are upgrading from a previous version, there are detailed instructions for how to upgrade safely.

0.3.13 - 2016/6/8

- Bug fix: LDAP auth disallows empty passwords for anonymous binding (pull 92)

0.3.12 - 2016/5/5

- Feature: Setting `auth.ldap.service_account` for LDAP auth (pull 84)

0.3.11 - 2016/4/28

- Bug fix: Missing newline in config template (pull 77)
- Feature: `pypi.always_show_upstream` for tweaking fallback behavior (issue 82)

0.3.10 - 2016/3/21

- Feature: S3 backend setting `storage.redirect_urls`

0.3.9 - 2016/3/13

- Bug fix: SQL cache works with MySQL (issue 74)
- Feature: S3 backend can use S3-compatible APIs (pull 72)

0.3.8 - 2016/3/10

- Feature: Cloudfront storage (pull 71)
- Bug fix: Rebuilding cache from storage won't crash on odd file names (pull 70)

0.3.7 - 2016/1/12

- Feature: `/packages` endpoint to list all files for all packages (pull 64)

0.3.6 - 2015/12/3

- Bug fix: Settings parsed incorrectly for LDAP auth (issue 62)

0.3.5 - 2015/11/15

- Bug fix: Mirror mode: only one package per version is displayed (issue 61)

0.3.4 - 2015/8/30

- Add docker-specific option for config creation
- Move docker config files to a separate repository

0.3.3 - 2015/7/17

- Feature: LDAP Support (pull 55)
- Bug fix: Incorrect package name/version when uploading from web (issue 56)

0.3.2 - 2015/7/7

- Bug fix: Restore direct links to S3 to fix easy_install (issue 54)

0.3.1 - 2015/6/18

- Bug fix: `pypi.allow_overwrite` causes crash in sql cache (issue 52)

0.3.0 - 2015/6/16

- Fully defines the behavior of every possible type of pip request. See Fallbacks for more detail.
- Don't bother caching generated S3 urls.

0.2.13 - 2015/5/27

- Bug fix: Crash when mirror mode serves private packages

0.2.12 - 2015/5/14

- Bug fix: Mirror mode works properly with S3 storage backend

0.2.11 - 2015/5/11

- Bug fix: Cache mode will correctly download packages with legacy versioning (pull 45)
- Bug fix: Fix the fetch_requirements endpoint (commit 6b2e2db)
- Bug fix: Incorrect expire time comparison with IAM roles (pull 47)
- Feature: 'mirror' mode. Caches packages, but lists all available upstream versions.

0.2.10 - 2015/2/27

- Bug fix: S3 download links expire incorrectly with IAM roles (issue 38)
- Bug fix: `fallback = cache` crashes with distlib 0.2.0 (issue 41)

0.2.9 - 2014/12/14

- Bug fix: Connection problems with new S3 regions (issue 39)
- Usability: Warn users trying to log in over http when `session.secure = true` (issue 40)

0.2.8 - 2014/11/11

- Bug fix: Crash when migrating packages from file storage to S3 storage (pull 35)

0.2.7 - 2014/10/2

- Bug fix: First download of package using S3 backend and `pypi.fallback = cache` returns 404 (issue 31)

0.2.6 - 2014/8/3

- Bug fix: Rebuilding SQL cache sometimes crashes (issue 29)

0.2.5 - 2014/6/9

- Bug fix: Rebuilding SQL cache sometimes deadlocks (pull 27)

0.2.4 - 2014/4/29

- Bug fix: `ppc-migrate` between two S3 backends (pull 22)

0.2.3 - 2014/3/13

- Bug fix: Caching works with S3 backend (commit 4dc593a)

0.2.2 - 2014/3/13

- Bug fix: Security bug in user auth (commit 001e8a5)
- Bug fix: Package caching from pypi was slightly broken (commit 065f6c5)
- Bug fix: `ppc-migrate` works when migrating to the same storage type (commit 45abcde)

0.2.1 - 2014/3/12

- Bug fix: Pre-existing S3 download links were broken by 0.2.0 (commit 52e3e6a)

0.2.0 - 2014/3/12

Upgrade breaks: caching database

- Bug fix: Timestamp display on web interface (pull 18)
- Bug fix: User registration stores password as plaintext (commit 21ebe44)
- Feature: `ppc-migrate`, command to move packages between storage backends (commit 399a990)
- Feature: Adding support for more than one package with the same version. Now you can upload wheels! (commit 2f24877)
- Feature: Allow transparently downloading and caching packages from pypi (commit e4dabc7)
- Feature: Export/Import access-control data via `ppc-export` and `ppc-import` (commit dbd2a16)
- Feature: Can set default read/write permissions for packages (commit c9aa57b)
- Feature: New cache backend: DynamoDB (commit d9d3092)

- Hosting all js & css ourselves (no more CDN links) (commit 20e345c)
- Obligatory miscellaneous refactoring

0.1.0 - 2014/1/20

- First public release

CHAPTER 2

API Reference

pypicloud package

Subpackages

pypicloud.access package

Submodules

pypicloud.access.base module

The access backend object base class

class pypicloud.access.base.**IAccessBackend**(*request=None*, *default_read=None*, *default_write=None*, *cache_update=None*)

Bases: object

Base class for retrieving user and package permission data

ROOT_ACL = [('Allow', 'system.Authenticated', 'login'), ('Allow', 'admin', <pyramid.security.AllPermissionsList object>)

allow_register()

Check if the backend allows registration

This should only be overridden by mutable backends

Returns allow : bool

allowed_permissions(*package*)

Get all allowed permissions for all principals on a package

Returns perms : dict

Mapping of principal to tuple of permissions

can_update_cache()
Return True if the user has permissions to update the pypi cache

classmethod configure(*settings*)
Configure the access backend with app settings

dump()
Dump all of the access control data to a universal format

Returns data : dict

get_acl(*package*)
Construct an ACL for a package

group_members(*group*)
Get a list of users that belong to a group

Parameters group : str

Returns users : list

List of user names

group_package_permissions(*group*)
Get a list of all packages that a group has permissions on

Parameters group : str

Returns packages : list

List of dicts. Each dict contains 'package' (str) and 'permissions' (list)

group_permissions(*package*, *group=None*)
Get a mapping of all groups to their permissions on a package

If a group is specified, just return the list of permissions for that group

Parameters package : str

The name of a python package

group : str, optional

The name of a single group the check

Returns permissions : dict

If group is None, mapping of group name to a list of permissions (which can contain 'read' and/or 'write')

permissions : list

If group is not None, a list of permissions for that group

Notes

You may specify special groups 'everyone' and/or 'authenticated', which correspond to all users and all logged in users respectively.

groups(*username=None*)
Get a list of all groups

If a username is specified, get all groups that the user belongs to

Parameters username : str, optional

Returns groups : list

List of group names

has_permission(*package*, *perm*)
Check if this user has a permission for a package

in_any_group(*username*, *groups*)
Find out if a user is in any of a set of groups

Parameters username : str

Name of user. May be None for the anonymous user.

groups : list

list of group names. Supports 'everyone', 'authenticated', and 'admin'.

Returns member : bool

in_group(*username*, *group*)
Find out if a user is in a group

Parameters username : str

Name of user. May be None for the anonymous user.

group : str

Name of the group. Supports 'everyone', 'authenticated', and 'admin'.

Returns member : bool

is_admin(*username*)
Check if the user is an admin

Parameters username : str

Returns is_admin : bool

load(*data*)
Idempotently load universal access control data.

By default, this does nothing on immutable backends. Backends may override this method to provide an implementation.

This method works by default on mutable backends with no override necessary.

mutable = False

need_admin()
Find out if there are any admin users

This should only be overridden by mutable backends

Returns need_admin : bool

True if no admin user exists and the backend is mutable, False otherwise

user_data(*username=None*)
Get a list of all users or data for a single user

For Mutable backends, this MUST exclude all pending users

Returns users : list

Each user is a dict with a 'username' str, and 'admin' bool

user : dict

If a username is passed in, instead return one user with the fields above plus a 'groups' list.

user_package_permissions(*username*)
Get a list of all packages that a user has permissions on

Parameters **username** : str

Returns **packages** : list

List of dicts. Each dict contains 'package' (str) and 'permissions' (list)

user_permissions(*package*, *username=None*)
Get a mapping of all users to their permissions for a package

If a username is specified, just return the list of permissions for that user

Parameters **package** : str

The name of a python package

username : str

The name of a single user the check

Returns **permissions** : dict

Mapping of username to a list of permissions (which can contain 'read' and/or 'write')

permissions : list

If username is not None, a list of permissions for that user

user_principals(*username*)
Get a list of principals for a user

Parameters **username** : str

Returns **principals** : list

verify_user(*username*, *password*)
Check the login credentials of a user

For Mutable backends, pending users should fail to verify

Parameters **username** : str

password : str

Returns **valid** : bool

True if user credentials are valid, false otherwise

class pypicloud.access.base.**IMutableAccessBackend**(*request=None*, *default_read=None*, *default_write=None*, *cache_update=None*)
Bases: *pypicloud.access.base.IAccessBackend*

Base class for access backends that can change user/group permissions

allow_register()

approve_user(*username*)
Mark a user as approved by the admin

Parameters **username** : str

create_group(*group*)
Create a new group

Parameters **group** : str

delete_group(*group*)
Delete a group

Parameters **group** : str

delete_user(*username*)
Delete a user

Parameters **username** : str

dump()

edit_group_permission(*package*, *group*, *perm*, *add*)
Grant or revoke a permission for a group on a package

Parameters **package** : str

group : str

perm : {'read', 'write'}

add : bool

If True, grant permissions. If False, revoke.

edit_user_group(*username*, *group*, *add*)
Add or remove a user to/from a group

Parameters **username** : str

group : str

add : bool

If True, add to group. If False, remove.

edit_user_password(*username*, *password*)
Change a user's password

Parameters **username** : str

password : str

edit_user_permission(*package*, *username*, *perm*, *add*)
Grant or revoke a permission for a user on a package

Parameters **package** : str

username : str

perm : {'read', 'write'}

add : bool

If True, grant permissions. If False, revoke.

load(*data*)

mutable = True

need_admin()

pending_users()
Retrieve a list of all users pending admin approval

Returns **users** : list

List of usernames

register(*username*, *password*)
Register a new user

The new user should be marked as pending admin approval

Parameters **username** : str

password : str

This should be the plaintext password

set_allow_register(*allow*)
Allow or disallow user registration

Parameters **allow** : bool

set_user_admin(*username*, *admin*)
Grant or revoke admin permissions for a user

Parameters **username** : str

admin : bool

If True, grant permissions. If False, revoke.

pypicloud.access.base.**group_to_principal**(*group*)
Convert a group to its corresponding principal

pypicloud.access.base.**groups_to_principals**(*groups*)
Convert a list of groups to a list of principals

pypicloud.access.config module

Backend that reads access control rules from config file

class pypicloud.access.config.**ConfigAccessBackend**(*request=None*, *settings=None*, *admins=None*, *group_map=None*, *user_groups=None*, ***kwargs*)
Bases: *pypicloud.access.base.IAccessBackend*

Access Backend that uses values set in the config file

classmethod **configure**(*settings*)

group_members(*group*)

group_package_permissions(*group*)

group_permissions(*package*, *group=None*)

groups(*username=None*)

is_admin(*username*)

load(*data*)

user_data(*username=None*)

user_package_permissions(*username*)

user_permissions(*package*, *username=None*)

pypicloud.access.ldap_ module

pypicloud.access.remote module

Backend that defers to another server for access control

class pypicloud.access.remote.**RemoteAccessBackend**(*request=None*, *settings=None*, *server=None*, *auth=None*, ***kwargs*)

Bases: *pypicloud.access.base.IAccessBackend*

This backend allows you to defer all user auth and permissions to a remote server. It requires the `requests` package.

classmethod configure(*settings*)

group_members(*group*)

group_package_permissions(*group*)

group_permissions(*package*, *group=None*)

groups(*username=None*)

is_admin(*username*)

user_data(*username=None*)

user_package_permissions(*username*)

user_permissions(*package*, *username=None*)

verify_user(*username*, *password*)

pypicloud.access.sql module

Access backend for storing permissions in using SQLAlchemy

class pypicloud.access.sql.**Group**(*name*)

Bases: sqlalchemy.ext.declarative.api.Base

Group record

name

class pypicloud.access.sql.**GroupPermission**(*package*, *groupname*, *read=False*, *write=False*)

Bases: *pypicloud.access.sql.Permission*

Permissions for a group on a package

group

groupname

package

read

write

class pypicloud.access.sql.**KeyVal**(*key*, *value*)

Bases: sqlalchemy.ext.declarative.api.Base

Simple model for storing key-value pairs

key

value

class pypicloud.access.sql.**Permission**(*package*, *read*, *write*)
Bases: sqlalchemy.ext.declarative.api.Base

Base class for user and group permissions

package = Column(None, String(length=255), table=None, primary_key=True, nullable=False)

permissions
Construct permissions list

read = Column(None, Boolean(), table=None)

write = Column(None, Boolean(), table=None)

class pypicloud.access.sql.**SQLAccessBackend**(*request=None*, *dbmaker=None*, ***kwargs*)
Bases: *pypicloud.access.base.IMutableAccessBackend*

This backend allows you to store all user and package permissions in a SQL database

allow_register()

approve_user(*username*)

classmethod configure(*settings*)

create_group(*group*)

delete_group(*group*)

delete_user(*username*)

edit_group_permission(*package*, *group*, *perm*, *add*)

edit_user_group(*username*, *groupname*, *add*)

edit_user_permission(*package*, *username*, *perm*, *add*)

group_members(*group*)

group_package_permissions(*group*)

group_permissions(*package*, *group=None*)

groups(*username=None*)

is_admin(*username*)

need_admin()

pending_users()

set_allow_register(*allow*)

set_user_admin(*username*, *admin*)

user_data(*username=None*)

user_package_permissions(*username*)

user_permissions(*package*, *username=None*)

class pypicloud.access.sql.**User**(*username*, *password*, *pending=True*)
Bases: sqlalchemy.ext.declarative.api.Base

User record

admin

groups

password

pending

username

class pypicloud.access.sql.**UserPermission**(*package*, *username*, *read=False*, *write=False*)
Bases: *pypicloud.access.sql.Permission*

Permissions for a user on a package

package

read

user

username

write

Module contents

Classes that provide user and package permissions

pypicloud.access.**includeme**(*config*)
Configure the app

pypicloud.cache package

Submodules

pypicloud.cache.base module

Base class for all cache implementations

class pypicloud.cache.base.**ICache**(*request=None*, *storage=None*, *allow_overwrite=None*)
Bases: object

Base class for a caching database that stores package metadata

all(*name*)
Search for all versions of a package

Parameters **name** : str

The name of the package

Returns **packages** : list

List of all *Package* s with the given name

clear(*package*)
Remove this package from the caching database

Parameters **package** : *Package*

clear_all()
Clear all cached packages from the database

classmethod **configure**(*settings*)
Configure the cache method with app settings

delete(*package*)
Delete this package from the database and from storage

Parameters **package** : *Package*

distinct()
Get all distinct package names

Returns **names** : list

List of package names

download_response(*package*)
Pass through to storage

fetch(*filename*)
Get matching package if it exists

Parameters **filename** : str

Name of the package file

Returns **package** : *Package*

get_url(*package*)
Get the download url for a package

Parameters **package** : *Package*

Returns **url** : str

package_class
alias of Package

reload_from_storage()
Make sure local database is populated with packages

reload_if_needed()
Reload packages from storage backend if cache is empty

This will be called when the server first starts

save(*package*)
Save this package to the database

Parameters **package** : *Package*

search(*criteria*, *query_type*)
Perform a search from pip

Parameters **criteria** : dict

Dictionary containing the search criteria. Pip sends search criteria for "name" and "summary" (typically, both of these lists have the same search values).

Example: {

"name": ["value1", "value2", ..., "valueN"], "summary": ["value1", "value2", ..., "valueN"]

}

query_type : str

Type of query to perform. By default, pip sends "or".

summary()
Summarize package metadata

Returns packages : list

List of package dicts, each of which contains 'name', 'stable', 'unstable', and 'last_modified'.

upload(*filename*, *data*, *name=None*, *version=None*, *summary=None*)
Save this package to the storage mechanism and to the cache

Parameters filename : str

Name of the package file

data : file

File-like readable object

name : str, optional

The name of the package (if not provided, will be parsed from filename)

version : str, optional

The version number of the package (if not provided, will be parsed from filename)

summary : str, optional

The summary of the package

Returns package : *Package*

The Package object that was uploaded

Raises e : ValueError

If the package already exists and allow_overwrite = False

pypicloud.cache.dynamo module

Store package data in DynamoDB

class pypicloud.cache.dynamo.**DynamoCache**(*request=None*, *engine=None*, ***kwargs*)
Bases: *pypicloud.cache.base.ICache*

Caching database that uses DynamoDB

all(*name*)

clear(*package*)

clear_all()

classmethod configure(*settings*)

distinct()

fetch(*filename*)

package_class
alias of *DynamoPackage*

save(*package*)

summary()

class pypicloud.cache.dynamo.**DynamoPackage**(*name*, *version*, *filename*, *last_modified=None*, *summary=None*, ***kwargs*)
Bases: *pypicloud.models.Package*, flywheel.models.Model

Python package stored in DynamoDB

data = <flywheel.fields.Field object>

filename = <flywheel.fields.Field object>

last_modified = <flywheel.fields.Field object>

meta_ = <flywheel.model_meta.ModelMetadata object>

name = <flywheel.fields.Field object>

summary = <flywheel.fields.Field object>

version = <flywheel.fields.Field object>

class pypicloud.cache.dynamo.**PackageSummary**(*package*)
Bases: flywheel.models.Model

Aggregate data about packages

last_modified = <flywheel.fields.Field object>

meta_ = <flywheel.model_meta.ModelMetadata object>

name = <flywheel.fields.Field object>

stable = <flywheel.fields.Field object>

unstable = <flywheel.fields.Field object>

update_with(*package*)
Update summary with a package

pypicloud.cache.redis_cache module

Store package data in redis

class pypicloud.cache.redis_cache.**RedisCache**(*request=None*, *db=None*, ***kwargs*)
Bases: *pypicloud.cache.base.ICache*

Caching database that uses redis

all(*name*)

clear(*package*)

clear_all()

classmethod configure(*settings*)

distinct()

fetch(*filename*)

redis_filename_set(*name*)
Get the key to a redis set of filenames for a package

redis_key(*key*)
Get the key to a redis hash that stores a package

redis_prefix = 'pypicloud:'

redis_set
Get the key to the redis set of package names

reload_from_storage()

save(*package*, *pipe=None*)

pypicloud.cache.sql module

Store package data in a SQL database

class pypicloud.cache.sql.**JSONEncodedDict**(**args*, ***kwargs*)
Bases: sqlalchemy.sql.type_api.TypeDecorator

Represents an immutable structure as a json-encoded string.

impl
alias of TEXT

process_bind_param(*value*, *dialect*)

process_result_value(*value*, *dialect*)

class pypicloud.cache.sql.**MutableDict**
Bases: sqlalchemy.ext.mutable.Mutable, dict

SQLAlchemy dict field that tracks changes

classmethod coerce(*key*, *value*)
Convert plain dictionaries to MutableDict.

class pypicloud.cache.sql.**SQLCache**(*request=None*, *dbmaker=None*, ***kwargs*)
Bases: *pypicloud.cache.base.ICache*

Caching database that uses SQLAlchemy

all(*name*)

clear(*package*)

clear_all()

classmethod configure(*settings*)

distinct()

fetch(*filename*)

package_class
alias of *SQLPackage*

reload_if_needed()

save(*package*)

search(*criteria*, *query_type*)
Perform a search.

Queries are performed as follows:

For the AND query_type, queries within a column will utilize the AND operator, but will not conflict with queries in another column.

(column1 LIKE '%a%' AND column1 LIKE '%b%') OR (column2 LIKE '%c%' AND column2 LIKE '%d%')

For the OR query_type, all queries will utilize the OR operator:

(column1 LIKE '%a%' OR column1 LIKE '%b%') OR (column2 LIKE '%c%' OR column2 LIKE '%d%')

summary()

class pypicloud.cache.sql.**SQLPackage**(*name*, *version*, *filename*, *last_modified=None*, *summary=None*, ***kwargs*)

Bases: *pypicloud.models.Package*, sqlalchemy.ext.declarative.api.Base

Python package stored in SQLAlchemy

data

filename

last_modified

name

summary

version

pypicloud.cache.sql.**create_schema**(*engine*)

Create the database schema if needed

Parameters **engine** : sqlalchemy.Engine

Notes

The method should only be called after importing all modules containing models which extend the Base object.

pypicloud.cache.sql.**drop_schema**(*engine*)

Drop the database schema

Parameters **engine** : sqlalchemy.Engine

Notes

The method should only be called after importing all modules containing models which extend the Base object.

Module contents

Caching database implementations

pypicloud.cache.**includeme**(*config*)

Get and configure the cache db wrapper

pypicloud.storage package

Submodules

pypicloud.storage.base module

Base class for storage backends

class pypicloud.storage.base.**IStorage**(*request*)
Bases: object

Base class for a backend that stores package files

classmethod configure(*settings*)
Configure the storage method with app settings

delete(*package*)
Delete a package file

Parameters **package** : *Package*

The package metadata

download_response(*package*)
Return a HTTP Response that will download this package

This is called from the download endpoint

get_url(*package*)
Create or return an HTTP url for a package file

By default this will return a link to the download endpoint

/api/package/<package>/<filename>

Returns **link** : str

Link to the location of this package file

list(*factory=<class 'pypicloud.models.Package'>*)
Return a list or generator of all packages

open(*package*)
Get a buffer object that can read the package data

This should be a context manager. It is used in migration scripts, not directly by the web application.

Parameters **package** : *Package*

Examples

```
with storage.open(package) as pkg_data:
    with open('outfile.tar.gz', 'w') as ofile:
        ofile.write(pkg_data.read())
```

upload(*package*, *data*)
Upload a package file to the storage backend

Parameters **package** : *Package*

The package metadata

data : file

A file-like object that contains the package data

pypicloud.storage.files module

Store packages as files on disk

class pypicloud.storage.files.**FileStorage**(*request=None*, ***kwargs*)
Bases: *pypicloud.storage.base.IStorage*

Stores package files on the filesystem

classmethod configure(*settings*)

delete(*package*)

download_response(*package*)

get_metadata_path(*package*)
Get the fully-qualified file path for a package metadata file

get_path(*package*)
Get the fully-qualified file path for a package

list(*factory=<class 'pypicloud.models.Package'>*)

open(*package*)

path_to_meta_path(*path*)
Construct the filename for a metadata file

upload(*package*, *data*)

pypicloud.storage.s3 module

Store packages in S3

class pypicloud.storage.s3.**CloudFrontS3Storage**(*request=None*, *bucket=None*, *expire_after=None*, *bucket_prefix=None*, *prepend_hash=None*, *cloud_front_domain=None*, *cloud_front_key_file=None*, *cloud_front_key_string=None*, *cloud_front_key_id=None*, ***kwargs*)
Bases: *pypicloud.storage.s3.S3Storage*

Storage backend that uses S3 and CloudFront

classmethod configure(*settings*)

class pypicloud.storage.s3.**S3Storage**(*request=None*, *bucket=None*, *expire_after=None*, *bucket_prefix=None*, *prepend_hash=None*, *redirect_urls=None*, *use_sse=False*, ***kwargs*)
Bases: *pypicloud.storage.base.IStorage*

Storage backend that uses S3

calculate_path(*package*)
Calculates the path of a package

classmethod configure(*settings*)

delete(*package*)

download_response(*package*)

get_path(*package*)
Get the fully-qualified bucket path for a package

get_url(*package*)

list(*factory=<class 'pypicloud.models.Package'>*)

open(**args*, ***kwds*)

test = False

upload(*package*, *data*)

Module contents

Storage backend implementations

pypicloud.storage.**get_storage_impl**(*settings*)
Get and configure the storage backend wrapper

pypicloud.views package

Submodules

pypicloud.views.admin module

API endpoints for admin controls

class pypicloud.views.admin.**AdminEndpoints**(*request*)
Bases: object

Collection of admin endpoints

approve_user()
Approve a pending user

create_group()
Create a group

delete_group()
Delete a group

delete_user()
Delete a user

download_access_control()
Download the ACL data as a gzipped-json file

edit_permission()
Edit user permission on a package

get_group()
Get the members and package permissions for a group

get_groups()
Get the list of groups

get_package_permissions()
Get the user and group permissions set on a package

get_pending_users()
Get the list of pending users

get_user()
Get a single user

get_user_permissions()
Get the package permissions for a user

get_users()
Get the list of users

mutate_group_member()
Add a user to a group

rebuild_package_list()
Rebuild the package cache in the database

set_admin_status(**args*, ***kwargs*)
Approve a pending user

toggle_allow_register(**args*, ***kwargs*)
Allow or disallow user registration

pypicloud.views.api module

Views for simple api calls that return json data

pypicloud.views.api.**all_packages**(**args*, ***kwargs*)
List all packages

pypicloud.views.api.**change_password**(**args*, ***kwargs*)
Change a user's password

pypicloud.views.api.**delete_package**(*context*, *request*)
Delete a package

pypicloud.views.api.**download_package**(*context*, *request*)
Download package, or redirect to the download link

pypicloud.views.api.**fetch_dist**(*request*, *package_name*, *package_url*)
Fetch a Distribution and upload it to the storage backend

pypicloud.views.api.**fetch_requirements**(**args*, ***kwargs*)
Fetch packages from the fallback_url

Parameters **requirements** : str

Requirements in the requirements.txt format (with newlines)

wheel : bool, optional

If True, will prefer wheels (default True)

prerelease : bool, optional

If True, will allow prerelease versions (default False)

Returns **pkgs** : list

List of Package objects

pypicloud.views.api.**package_versions**(**args*, ***kwargs*)
List all unique package versions

pypicloud.views.api.**register**(**args*, ***kwargs*)
Register a user

pypicloud.views.api.**upload_package**(**args*, ***kwargs*)
Upload a package

pypicloud.views.login module

Render views for logging in and out of the web interface

pypicloud.views.login.**do_forbidden**(*request*)
Intercept 403's and return 401's when necessary

pypicloud.views.login.**do_login**(**args*, ***kwargs*)
Check credentials and log in

pypicloud.views.login.**get_login_page**(*request*)
Catch login and redirect to login wall

pypicloud.views.login.**handle_register_request**(*request*, *username*, *password*)
Process a request to register a new user

pypicloud.views.login.**logout**(*request*)
Delete the user session

pypicloud.views.login.**register**(**args*, ***kwargs*)
Check credentials and log in

pypicloud.views.login.**register_new_user**(*access*, *username*, *password*)
Register a new user & handle duplicate detection

pypicloud.views.packages module

View for cleaner buildout calls

pypicloud.views.packages.**list_packages**(**args*, ***kwargs*)
Render the list for all versions of all packages

pypicloud.views.simple module

Views for simple pip interaction

pypicloud.views.simple.**get_fallback_packages**(*request*, *package_name*, *redirect=True*)
Get all package versions for a package from the fallback_url

pypicloud.views.simple.**package_versions**(**args*, ***kwargs*)
Render the links for all versions of a package

pypicloud.views.simple.**packages_to_dict**(*request*, *packages*)
Convert a list of packages to a dict used by the template

pypicloud.views.simple.**search**(*request*, *criteria*, *query_type*)
Perform searches from pip. This handles XML RPC requests to the "pypi" endpoint (configured as /pypi/) that specify the method "search".

pypicloud.views.simple.**simple**(**args*, ***kwargs*)
Render the list of all unique package names

pypicloud.views.simple.**upload**(**args*, ***kwargs*)
Handle update commands

Module contents

Views

pypicloud.views.**get_index**(**args*, ***kwargs*)
Render a home screen

pypicloud.views.**health_endpoint**(*request*)
Simple health endpoint

Submodules

pypicloud.auth module

Utilities for authentication and authorization

class pypicloud.auth.**BasicAuthenticationPolicy**
Bases: object

A **:app:'Pyramid'** authentication policy which obtains data from basic authentication headers.

Constructor Arguments

check

A callback passed the credentials and the request, expected to return None if the userid doesn't exist or a sequence of group identifiers (possibly empty) if the user does exist. Required.

authenticated_userid(*request*)
Verify login and return the authed userid

effective_principals(*request*)
Get the authed groups for the active user

forget(*request*)
HTTP headers to forget credentials

remember(*request*, *principal*, ***kw*)
HTTP Headers to remember credentials

unauthenticated_userid(*request*)
Return userid without performing auth

class pypicloud.auth.**SessionAuthPolicy**
Bases: object

Simple auth policy using beaker sessions

authenticated_userid(*request*)
Return the authenticated userid or None if no authenticated userid can be found. This method of the policy should ensure that a record exists in whatever persistent store is used related to the user (the user should not have been deleted); if a record associated with the current id does not exist in a persistent store, it should return None.

effective_principals(*request*)
Return a sequence representing the effective principals including the userid and any groups belonged to by

the current user, including 'system' groups such as `pyramid.security.Everyone` and `pyramid.security.Authenticated`.

forget(*request*)
Return a set of headers suitable for 'forgetting' the current user on subsequent requests.

remember(*request*, *principal*, **_)
This implementation is slightly different than expected. The application should call remember(userid) rather than remember(principal)

unauthenticated_userid(*request*)
Return the *unauthenticated* userid. This method performs the same duty as `authenticated_userid` but is permitted to return the userid based only on data present in the request; it needn't (and shouldn't) check any persistent store to ensure that the user record related to the request userid exists.

pypicloud.auth.**get_basicauth_credentials**(*request*)
Get the user/password from HTTP basic auth

pypicloud.auth.**includeme**(*config*)
Configure the app

pypicloud.models module

Model objects

class pypicloud.models.**Package**(*name*, *version*, *filename*, *last_modified=None*, *summary=None*, ***kwargs*)
Bases: object

Representation of a versioned package

Parameters **name** : str

The name of the package (will be normalized)

version : str

The version number of the package

filename : str

The name of the package file

last_modified : datetime, optional

The datetime when this package was uploaded (default now)

summary : str, optional

The summary of the package

****kwargs** : dict

Metadata about the package

get_url(*request*)
Create path to the download link

is_prerelease
Returns True if the version is a prerelease version

parsed_version
Parse and cache the version using pkg_resources

search_summary()
Data to return from a pip search

pypicloud.route module

Tools and resources for traversal routing

class pypicloud.route.**APIPackageFileResource**(*request*, *name*, *filename*)
Bases: object

Resource for api endpoints dealing with a single package version

class pypicloud.route.**APIPackageResource**(*request*, *name*)
Bases: *pypicloud.route.IResourceFactory*

Resource for requesting package versions

class pypicloud.route.**APIPackagingResource**(*request*)
Bases: *pypicloud.route.IResourceFactory*

Resource for api package queries

class pypicloud.route.**APIResource**(*request*)
Bases: *pypicloud.route.IStaticResource*

Resource for api calls

subobjects = {'package': <class 'pypicloud.route.APIPackagingResource'>}

class pypicloud.route.**AdminResource**(*request*)
Bases: *pypicloud.route.IStaticResource*

Resource for admin calls

class pypicloud.route.**IResourceFactory**(*request*)
Bases: object

Resource that generates child resources from a factory

class pypicloud.route.**IStaticResource**(*request*)
Bases: object

Simple resource base class for static-mapping of paths

subobjects = {}

class pypicloud.route.**PackagesResource**(*request*)
Bases: *pypicloud.route.IStaticResource*

Resource for cleaner buildout config

class pypicloud.route.**Root**(*request*)
Bases: *pypicloud.route.IStaticResource*

Root context for PyPI Cloud

subobjects = {'admin': <class 'pypicloud.route.AdminResource'>, 'simple': <class 'pypicloud.route.SimpleResource'>

class pypicloud.route.**SimplePackageResource**(*request*, *name*)
Bases: object

Resource for requesting simple endpoint package versions

class pypicloud.route.**SimpleResource**(*request*)
Bases: object

Resource for simple pip calls

pypicloud.scripts module

Commandline scripts

pypicloud.scripts.**export_access**(*argv=None*)
Dump the access control data to a universal format

pypicloud.scripts.**gen_password**()
Generate a salted password

pypicloud.scripts.**import_access**(*argv=None*)
Load the access control data from a dump file or stdin

This operation is idempotent and graceful. It will not clobber your existing ACL.

pypicloud.scripts.**make_config**(*argv=None*)
Create a server config file

pypicloud.scripts.**migrate_packages**(*argv=None*)
Migrate packages from one storage backend to another

Create two config.ini files that are configured to use different storage backends. All packages will be migrated from the storage backend in the first to the storage backend in the second.

ex: pypicloud-migrate-packages file_config.ini s3_config.ini

pypicloud.scripts.**prompt**(*msg*, *default=<object object>*, *validate=None*)
Prompt user for input

pypicloud.scripts.**prompt_option**(*text*, *choices*, *default=<object object>*)
Prompt the user to choose one of a list of options

pypicloud.scripts.**promptyn**(*msg*, *default=None*)
Display a blocking prompt until the user confirms

pypicloud.util module

Utilities

class pypicloud.util.**BetterScrapingLocator**(**args*, ***kw*)
Bases: distlib.locators.SimpleScrapingLocator

Layer on top of SimpleScrapingLocator that allows preferring wheels

locate(*requirement*, *prereleases=False*, *wheel=True*)

prefer_wheel = True

score_url(*url*)

class pypicloud.util.**NormalizeNameHackString**
Bases: unicode

Super hacked wrapper around a string that runs normalize_name before doing equality comparisons

lower()

pypicloud.util.**create_matcher**(*queries*, *query_type*)
Create a matcher for a list of queries

Parameters **queries** : list

List of queries

query_type: str

Type of query to run: ["or"|"and"]

Returns Matcher function

pypicloud.util.**getdefaults**(*settings*, **args*)
Attempt multiple gets from a dict, returning a default value if none of the keys are found.

pypicloud.util.**is_compatible**(*wheel*, *tags=None*)
Hacked function to monkey patch into distlib

pypicloud.util.**normalize_name**(*name*)
Normalize a python package name

pypicloud.util.**parse_filename**(*filename*, *name=None*)
Parse a name and version out of a filename

Module contents

S3-backed pypi server

pypicloud.**hook_exceptions**()
Hooks into the sys module to set our formatter.

pypicloud.**includeme**(*config*)
Set up and configure the pypicloud app

pypicloud.**main**(*config*, ***settings*)
This function returns a Pyramid WSGI application.

pypicloud.**to_json**(*value*)
A json filter for jinja2

pypicloud.**traceback_formatter**(*excpt*, *value*, *tback*)
Catches all exceptions and re-formats the traceback raised.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

D

E

F

G

H

R

S

T

U

V

W