twitcher Documentation

Release 0.7.0

Birdhouse

May 11, 2022

CONTENTS

1 Overview 3

2 Installation 5

3 Configuration 7

4 Developer Guide 9

5 Tutorial 13

6 API Reference 19

7 Changes 27

Python Module Index 35

Index 37

Twitcher (the bird-watcher) *a birdwatcher mainly interested in catching sight of rare birds.* (Leo).

Twitcher is a security proxy for OWS services like Web Processing Services (WPS). The proxy service uses OAuth2 access tokens to protect the OWS service access. In addition one can also use X.509 certificates for client authentication.

The implementation is not restricted to WPS services. It will be extended to more OWS services like WMS (Web Map Service) and might also be used for Thredds catalog services.

Twitcher extensions:

- Magpie is an AuthN/AuthZ service provided by the PAVICS project.
- Weaver middleware by CRIM. A reimplementation of an old Twitcher fork for workflow execution and a Swagger RESTful interface for Web Processing Services.

Twitcher is implemented with the Python Pyramid web framework.

You can try Twitcher online using Binder, or view the notebooks on NBViewer.

Twitcher is part of the Birdhouse project. The documentation is on ReadTheDocs.

Twitcher Docker images are also available for most recent tagged versions.

CHAPTER

ONE

OVERVIEW

1.1 Twitcher Components

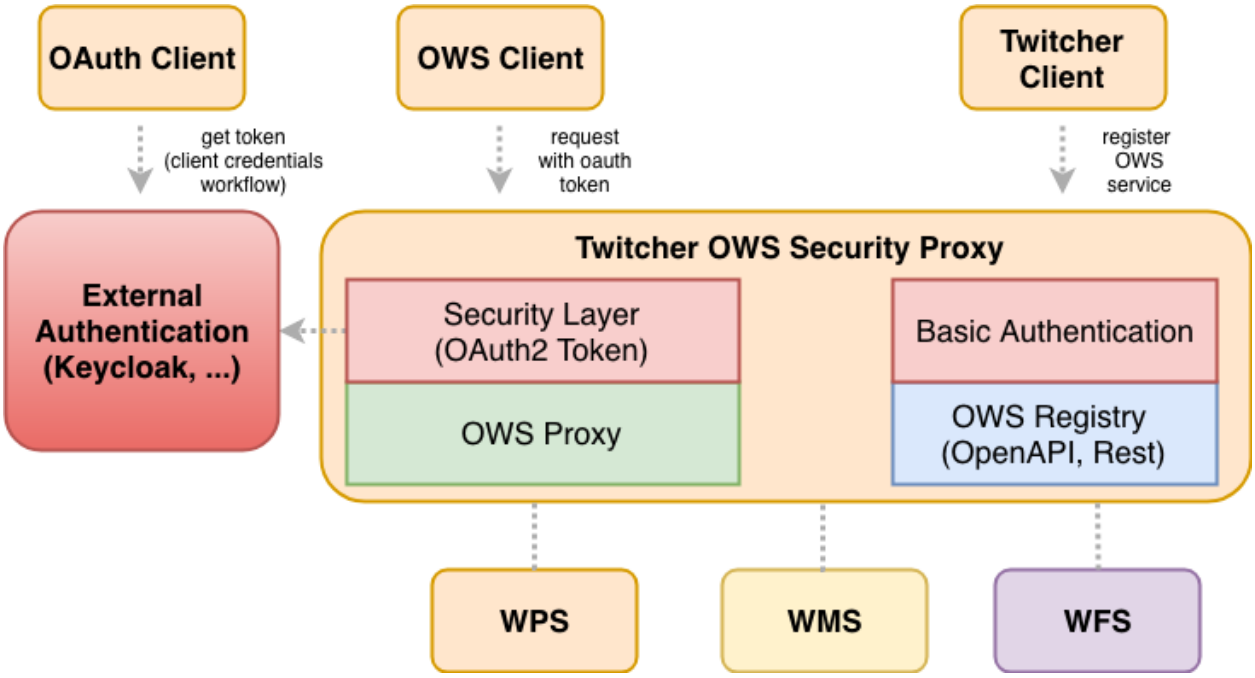
Twitcher consists of the following main parts:

OWS Security A security layer to protect service access with *OAuth2 Tokens*.

OWS Registry *OWS Registry* is a registration service with an *OpenAPI interface* to register OWS services for the OWS proxy protected by basic authentication.

OWS Proxy *OWS Proxy* a service which acts as a proxy for registered OWS services.

OAuth access tokens can be retrieved from a Keycloak authentication service using the client credentials workflow.



CHAPTER

TWO

INSTALLATION

2.1 Quick Installation

Quick steps to install and start Twitcher:

```
$ git clone https://github.com/bird-house/twitcher.git
$ cd twitcher
$ conda env create
$ conda activate twitcher
$ make install
$ make migrate
$ make start
```

2.2 From GitHub Sources

Get Twitcher source from GitHub:

```
$ git clone https://github.com/bird-house/twitcher.git
$ cd twitcher
```

Create Conda environment named *twitcher*:

```
$ conda env update -f environment.yml
$ conda activate twitcher
```

Install the Twitcher app:

```
$ pip install -e .
OR
make install
```

For development you can use this command:

```
$ pip install -e .[dev]
OR
$ make develop
```

2.3 Initialize Database

Before you can start the service you need to initialize or upgrade the database:

```
$ make migrate
```

2.4 Starting Twitcher Service

For development twitcher is using the the waitress WSGI server.

Start the twitcher service using the *development.ini* configuration:

```
$ pserve development.ini --reload
OR
$ make start
```

CHAPTER

THREE

CONFIGURATION

Twitcher has a configuration file *development.ini* for local development. Copy and edit this configuration to adapt to your settings.

3.1 Service

Edit the configuration to change the service parameters.

The URL of the Twitcher service endpoint:

```
twitcher.url = http://localhost:8000
```

3.2 Basic Authentication

Twitcher uses basic authentication for client application registration and service registration. Edit username and password in the configuration:

```
twitcher.username = demo
twitcher.password = demo
```

3.3 OAuth2 Token Generator

Twitcher uses OAuth2 tokens to control OWS service access. You can use several types of tokens.

3.3.1 Random Token

Tokens with UUID strings stored in the local twitcher database.

Edit the configuration file:

```
twitcher.token.type = random_token
```

3.3.2 Signed Token

JWT tokens signed with a certificate. You can generate a self-signed certificate for testing with the Makefile:

```
$ make gencert
```

Edit the configuration file:

```
twitcher.token.type = signed_token
twitcher.token.keyfile = key.pem # private key
twitcher.token.certfile = pubkey.pem # public key
```

3.3.3 Custom Token

JWT tokens using a shared secret. You can generate a UUID secret with:

```
$ make gensecret
```

Edit the configuration file:

```
twitcher.token.type = custom_token
twitcher.token.secret = secret
```

3.3.4 Keycloak Token

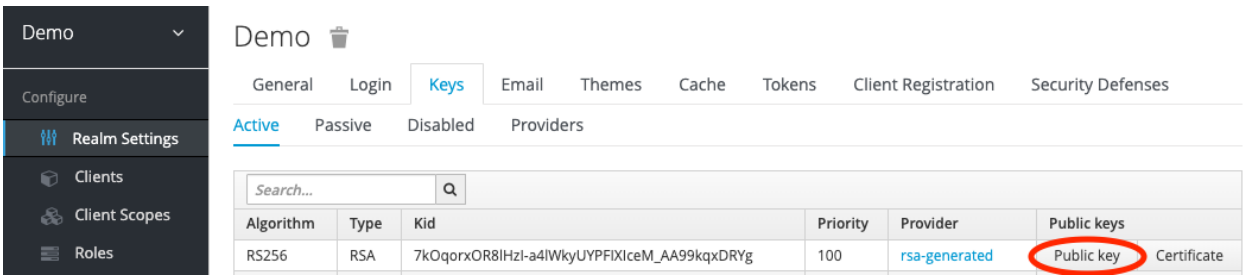
JWT tokens generated by a Keycloak OAuth2 service.

Edit the configuration file:

```
twitcher.token.type = keycloak_token
```

You need to copy the public key of your Keycloak realm to the configuration (see screenshot):

```
keycloak.token.secret = secret
```



CHAPTER

FOUR

DEVELOPER GUIDE

- *Running tests*
- *Run tests the lazy way*
- *Upgrade Database*
- *Building the docs*
- *Prepare a release*
- *Bump a new version*

4.1 Running tests

Run tests using pytest.

First activate the `twitcher` Conda environment and install `pytest`.

```
$ source activate twitcher
$ pip install -r requirements_dev.txt  # if not already installed
OR
$ make develop
```

Run quick tests (skip slow and online):

```
$ pytest -m 'not slow and not online'"
```

Run all tests:

```
$ pytest
```

Check pep8:

```
$ flake8
```

4.2 Run tests the lazy way

Do the same as above using the `Makefile`.

```
$ make test
$ make test-all
$ make lint
$ make coverage
```

4.3 Upgrade Database

Initialize and upgrade the database using Alembic.

Generate your first revision:

```
$ conda activate twitcher
$ alembic -c development.ini revision --autogenerate -m "init"
```

Warning: This first step is only needed in development to generate a new database schema version.

Upgrade to that revision:

```
$ alembic -c development.ini upgrade head
```

Load default data into the database using a script.

```
$ initialize_twitcher_db development.ini
```

Note: You can use *make migrate* as a shortcut to upgrade or init the twitcher database (last two steps).

4.4 Building the docs

First install dependencies for the documentation:

```
$ make docs
```

4.5 Prepare a release

Update *CHANGES.rst*.

4.6 Bump a new version

Make a new version of twitcher in the following steps:

- Make sure everything is commit to GitHub.
- Update `CHANGES.rst` with the next version.
- Dry Run: `bumpversion --dry-run --verbose --new-version 0.5.1 patch`
- Do it: `bumpversion --new-version 0.5.1 patch`
- ... or: `bumpversion --new-version 0.6.0 minor`
- Push it: `git push`
- Push tag: `git push --tags`

See the bumpversion documentation for details.

CHAPTER

FIVE

TUTORIAL

- *Using the OWSProxy with a WPS application*
  - *Prepare your Client Application*
  - *Register a WPS service*
  - *Access a registered service*
  - *Use a token to run an execute request*
- *Use x509 certificates to control client access*
- *Keycloak example*

5.1 Using the OWSProxy with a WPS application

The `OWSProxy` is a proxy service for OWS services.

First you need an external WPS. You can use Emu WPS service from Birdhouse. Get it from GitHub and run the installation:

```
$ git clone https://github.com/bird-house/emu.git
$ cd emu
$ make install
$ make start
```

The Emu WPS service is available by default at the URL: http://localhost:5000/wps?service=WPS&version=1.0.0&request=GetCapabilities

Make sure Twitcher is installed and running:

```
$ cd ../twitcher  # cd into the twitcher installation folder
$ pserve development.ini
```

5.1.1 Prepare your Client Application

Register your client application at twitcher to get a *client_id* and *client_secret*:

```
$ twitcherctl -k --username demo --password demo add --name demo_app
{'name': 'demo_app', 'client_id': 'id', 'client_secret': 'secret'}
```

Get an access token to use the registration service using your OAuth *client_id* and *client_secret* with scope *register*:

```
$ twitcherctl -k gentoken -i client_id -s client_secret --scope register
{'access_token': 'TOKEN', 'expires_in': 3600, 'scope': ['register'], 'token_type':
↪'Bearer'}
```

5.1.2 Register a WPS service

Register the Emu WPS service at the Twitcher `OWSProxy`:

```
$ twitcherctl -k --username demo --password demo register --name emu http://localhost:
↪5000/wps
```

If you don't provide a name with `--name` option then a nice name will be generated, for example `sleepy_flamingo`.

Use the `list` command to see which WPS services are registered with OWSProxy:

```
$ twitcherctl -k --username demo --password demo list
[{'url': 'http://localhost:5000/wps', 'type': 'wps', 'name': 'emu', 'auth': 'token'}]
```

5.1.3 Access a registered service

By default the registered service is available at the URL `https://localhost:8000/ows/proxy/{service_name}`. Replace the `service_name` with the registered name.

Run a `GetCapabilities` request for the registered Emu WPS service:

```
$ curl -k "http://localhost:8000/ows/proxy/emu?service=WPS&request=GetCapabilities"
```

Run a `DescribeProcess` request:

```
$ curl -k "http://localhost:8000/ows/proxy/emu?service=WPS&version=1.0.0&
↪request=DescribeProcess&identifier=hello"
```

5.1.4 Use a token to run an execute request

By default the WPS service is protected by the `OWSSecurity` wsgi middleware. You need to provide an OAuth access token to run an execute request.

Run an `Exceute` request:

```
$ curl -k "http://localhost:8000/ows/proxy/emu?service=WPS&version=1.0.0&request=Execute&
↪identifier=hello&DataInputs=name=tux"
```

Now you should get an XML error response with a message that you need to provide an access token.

We need to generate an access token with `twitcherctl` using OAuth *client_id* and *client_secret* with scope *compute*:

```
$ twitcherctl -k gentoken -i client_id -s client_secret --scope compute
{'access_token': 'TOKEN', 'expires_in': 3600, 'scope': ['compute'], 'token_type': 'Bearer
↪'}
```

By default the token has a limited life time of one hour.

For testing you can provide the OAuth token as HTTP parameter:

```
$ curl -k "http://localhost:8000/ows/proxy/emu?service=WPS&version=1.0.0&request=Execute&
↪identifier=hello&DataInputs=name=tux&access_token=TOKEN"
```

But you should use an HTTP header:

```
$ curl -k -H 'Authorization: Bearer TOKEN' "http://localhost:8000/ows/proxy/emu?
↪service=WPS&version=1.0.0&request=Execute&identifier=hello&DataInputs=name=tux"
```

5.2 Use x509 certificates to control client access

Warning: You need an Nginx web-server in front of the Twitcher WSGI service to use x509 certificates.

Hint: You can install Twitcher with Nginx using an Ansible **playbook_**.

Since version 0.3.6 Twitcher is prepared to use x509 certificates to control client access. By default it is configured to accept x509 proxy certificates from ESGF.

Register the Emu WPS service at the Twitcher `OWSProxy` with `auth` option `cert`:

```
$ twitcherctl -k --username demo --password demo register --name emu --auth cert http://
↪localhost:5000/wps
```

The `GetCapabilities` and `DescribeProcess` requests are not blocked:

```
$ curl -k "http://localhost:8000/ows/proxy/emu?service=WPS&request=GetCapabilities"
$ curl -k "http://localhost:8000/ows/proxy/emu?service=WPS&version=1.0.0&
↪request=DescribeProcess&identifier=hello"
```

When you run an `Exceute` request without a certificate you should get an exception report:

```
$ curl -k "http://localhost:8000/ows/proxy/emu?service=WPS&version=1.0.0&request=Execute&
↪identifier=hello&DataInputs=name=tux"
```

Now you should get an XML error response with a message that you need to provide a valid X509 certificate.

Get a valid proxy certificate from ESGF, you may use the esgf-pyclient to run a myproxy logon. Let's say your proxy certificate is `cert.pem`, then run the exceute request again using this certificate:

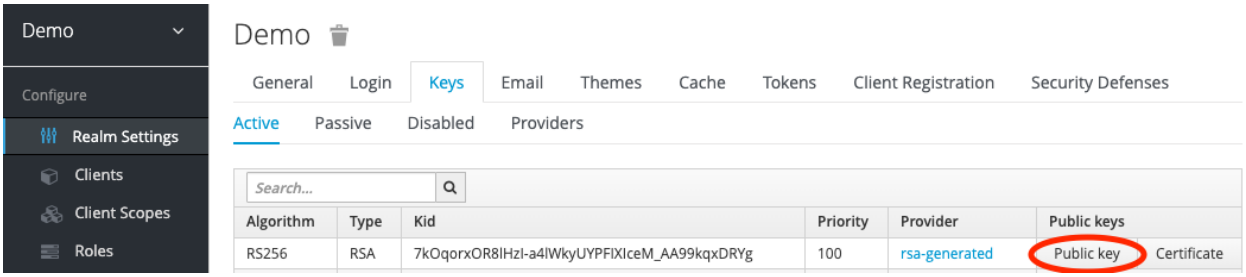
```
$ curl --cert cert.pem --key cert.pem -k "http://localhost:8000/ows/proxy/emu?
↪service=WPS&version=1.0.0request=Execute&identifier=hello&DataInputs=name=tux"
```

5.3 Keycloak example

Set-up a demo Keycloak service using an Ansible playbook.

The keycloak service is available at (`username=admin`, `password=admin`): http://localhost:8080/auth/

You need to copy the public key of your Keycloak realm to the twitcher configuration (see screenshot):



Update your twitcher configuration in `development.ini`:

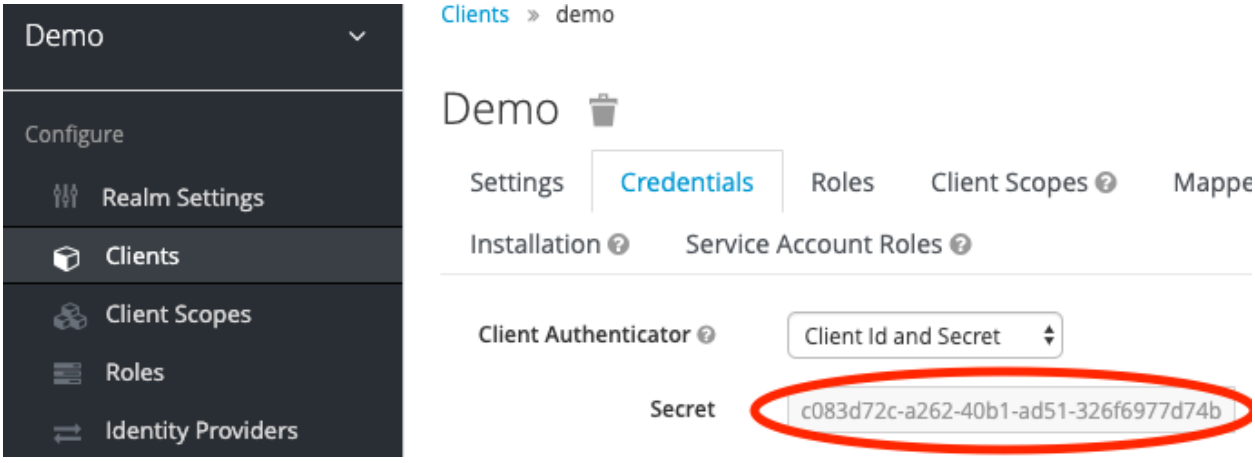
```
twitcher.token.type = keycloak_token
keycloak.token.secret = public_key_copied_from_keycloak
```

Start the twitcher service and register the Emu WPS:

```
$ twitcherctl -k --username demo --password demo register --name emu http://localhost:
↪5000/wps
```

Try the demo notebook to access a token from the keycloak and execute a WPS process.

Use `client_id=demo` and copy the client secret from Keycloak in *Clients/demo/Credentials/Secret* (see screenshot).



5.3.1 Using Twitcher with Keycloak

Setup a Keycloak service with client credentials grant type. You can use this Ansible playbook. The Keycloak service is running on: http://localhost:8080

Configure the twitcher as described in the keycloak example of the twitcher tutorial and register the Emu WPS.

[ ]:
```
# disable ssl warnings
import urllib3
urllib3.disable_warnings()
```

Keycloak client

https://www.keycloak.org/docs/latest/server_admin/index.html#_service_accounts

[ ]:
```
keycloak_url = 'http://localhost:8080'
token_endpoint = '/auth/realms/demo/protocol/openid-connect/token'
client_id = 'demo'
client_secret = 'c083d72c-a262-40b1-ad51-326f6977d74b'
```

Get OAuth access token from Keycloak

scope=compute

[ ]:
```
token_url = "{}{}".format(keycloak_url, token_endpoint)
token_url
```

[ ]:
```
import os
from oauthlib.oauth2 import BackendApplicationClient
from requests_oauthlib import OAuth2Session

os.environ['OAUTHLIB_INSECURE_TRANSPORT'] = '1'

client = BackendApplicationClient(client_id=client_id)
oauth = OAuth2Session(client=client)
token = oauth.fetch_token(
    token_url,
    scope='compute',
    client_id=client_id,
    client_secret=client_secret,
    include_client_id=True,
    verify=False)
token
```

[ ]:
```
token['access_token']
```

Execute WPS Process with access token

[ ]:
```
base_url = 'http://localhost:8000'
url = "{}/ows/proxy/emu?service=WPS&version=1.0.0&request=Execute&identifier=chomsky".
↪format(base_url)
url
```

[ ]:
```
import requests
headers = {'Authorization': 'Bearer {}'.format(token['access_token'])}

resp = requests.get(url, headers=headers, verify=False)
resp.ok
```

[ ]:
```
'ProcessSucceeded' in resp.text
```

CHAPTER

SIX

API REFERENCE

- *Twitcher Client*
- *Twitcher CLI*
- *OpenAPI interface*
- *OAuth2 Tokens*
- *OWS Security*
- *OWS Registry*
- *OWS Proxy*

6.1 Twitcher Client

class twitcher.client.**TwitcherService**(*url*, *username=None*, *password=None*, *verify=True*)

TwitcherService is a twitcher client to talk to the twitcher service API.

add_client_app(*name=None*, *redirect_uri=None*)

Add a client application to twitcher with optional name.

clear_services()

Remove all OWS services.

fetch_token(*client_id*, *client_secret*, *scope=None*, *keycloak=False*)

Get an access token with given scope.

get_service(*name*)

Get an OWS service with given name.

list_services()

List all registered OWS services.

register_service(*name*, *url*, *data=None*)

Register a service.

unregister_service(*name*)

Remove registered service with given name.

6.2 Twitcher CLI

The `twitcherctl` is a command line tool to control the twitcher service. It is used to generate access tokens and to register OWS services.

`twitcherctl` is part of the twitcher installation:

```
$ twitcherctl -h
```

6.2.1 *twitcherctl* Commands and Options

`twitcherctl` has the following command line options:

-h, --help Print usage message and exit

-s, --serverurl URL on which twitcher server is listening (default "http://localhost:8000/").

-u, --username Username to access twitcher server.

-p, --password Password to access twitcher server.

-k, --insecure Don't validate the server's certificate.

List of available commands:

add Add OAuth2 client application.

gentoken Generates an access token.

list Lists all registered OWS services used by OWS proxy.

clear Removes all OWS services from the registry.

register Adds OWS service to the registry to be used by the OWS proxy.

unregister Removes OWS service from the registry.

6.2.2 Add an OAuth2 client application

Register an application using basic authentication with username and password (name and redirect-uri are optional):

```
$ twitcherctl -k --username demo --password demo add --name demo_app --redirect-uri http:
↪//localhost/demo_app
{'client_id': 'id', 'client_secret': 'secret'}
```

The result is an OAuth *client_id* and *client_secret*.

6.2.3 Generate an access token

Get an OAuth access token using *client_id* and *client_secret* for given *scope*:

```
$ twitcherctl -k gentoken -i client_id -s client_secret --scope compute
{'access_token': 'TOKEN', 'expires_in': 3600, 'scope': ['compute'], 'token_type': 'Bearer
↪'}
```

Possible scopes are: compute, register.

You can also get a token from a Keycloak OAuth service using the client credentials workflow:

```
$ twitcherctl -k -s http://localhost:8080 gentoken -i client_id -s client_secret --scope␣
↪compute --keycloak
```

6.2.4 Register an OWS Service for the OWS Proxy

See the available options:

```
twitcherctl -k register -h
```

Register a local WPS service using an OAuth access token:

```
$ twitcherctl -k --username demo --password demo register http://localhost:5000/wps
tiny_buzzard
```

You can use the `--name` option to provide a name (used by the OWS proxy). Otherwise a nice name will be generated.

6.2.5 List registered services

The `list` command shows the registered OWS services:

```
$ twitcherctl -k --username demo --password demo list
[{'url': 'http://localhost:5000/wps', 'type': 'wps', 'name': 'tiny_buzzard', 'auth':
↪'token'}]
```

class twitcher.scripts.twitcherctl.**TwitcherCtl**

Command line to interact with the OAuth and OpenAPI interface of the `twitcher` service.

6.3 OpenAPI interface

class twitcher.api.**TwitcherAPI**

Twitcher API defined with OpenAPI.

static clear_services(*request*)

Clear all services.

static get_service(*request*)

Get registered service.

static list_services(*request*)

Returns a list of registered services.

static register_service(*request*)

Register a service.

static unregister_service(*request*)

Remove registered service.

6.4 OAuth2 Tokens

6.4.1 twitcher.oauth2

This module implements the OAuth2 model to generate access tokens. These tokens are used to access the compute services (*scope=compute*). The compute services are accessed via the OWS proxy.

The implementation is using the OAuth2 client credentials grant type

Currently three types of access tokens can be used:

random_token The access token is a UUID string. The tokens are stored in a local database and can be used for local validation only.

signed_token A JWT token signed with a X.509 certificate. The token can be validated without contacting a validation service.

custom_token A JWT token with a secret (UUID string) which can be shared for validation. The token can be validated without contacting a validation service.

keycloak_token A JWT token generated by a Keycloak OAuth2 service.

See also the OAuth2 token documenation

The implementation is using the pyramid-oauthlib library. The code is also inspired by the following OAuth libraries:

- https://github.com/thomsonreuters/bottle-oauthlib
- https://github.com/lepture/flask-oauthlib

Further reading:

- https://oauthlib.readthedocs.io/en/latest/index.html
- https://pypi.org/project/PyJWT/
- https://requests-oauthlib.readthedocs.io/en/latest/index.html
- https://www.slideshare.net/alvarosanchezmariscal/stateless-authentication-with-oauth-2-and-jwt-javazone-2015
- https://docs.apigee.com/api-platform/security/oauth/oauth-20-client-credentials-grant-type

`twitcher.oauth2.`**`generate_token_view`**(*request*)

Core functionality is available directly from the request.

Responses from OAuthLib are wrapped in a response object of type `pyramid.response.Response` so they can be returned directly from views.

`twitcher.oauth2.`**`register_client_app_view`**(*request*)

Register a new client application and returns `client_id` and `client_secret`.

Uses basic authentication.

`class` `twitcher.oauth2.`**`RandomTokenValidator`**

`save_bearer_token`(*token_response*, *request*, **args*, ***kwargs*)

Persist the Bearer token.

`validate_bearer_token`(*token*, *scopes*, *request*)

Validate access token.

Parameters

- **`token`** – A string of random characters

- **scopes** – A list of scopes
- **request** – The Request object passed by oauthlib

The validation validates:

1) if the token is available
2) if the token has expired
3) if the scopes are available

class twitcher.oauth2.**SignedTokenValidator**(*cert*, *key*, *issuer*)

validate_bearer_token(*token*, *scopes*, *request*)

Ensure the Bearer token is valid and authorized access to scopes.

Parameters

- **token** – A string of random characters.
- **scopes** – A list of scopes associated with the protected resource.
- **request** – The HTTP Request (oauthlib.common.Request)

A key to OAuth 2 security and restricting impact of leaked tokens is the short expiration time of tokens, *always ensure the token has not expired!*.

Two different approaches to scope validation:

1) **all(scopes). The token must be authorized access to all scopes** associated with the resource. For example, the token has access to `read-only` and `images`, thus the client can view images but not upload new. Allows for fine grained access control through combining various scopes.
2) **any(scopes). The token must be authorized access to one of the** scopes associated with the resource. For example, token has access to `read-only-images`. Allows for fine grained, although arguably less convenient, access control.

A powerful way to use scopes would mimic UNIX ACLs and see a scope as a group with certain privileges. For a restful API these might map to HTTP verbs instead of read, write and execute.

Note, the request.user attribute can be set to the resource owner associated with this token. Similarly the request.client and request.scopes attribute can be set to associated client object and authorized scopes. If you then use a decorator such as the one provided for django these attributes will be made available in all protected views as keyword arguments.

Parameters

- **token** – Unicode Bearer token
- **scopes** – List of scopes (defined by you)
- **request** – The HTTP Request (oauthlib.common.Request)

Return type True or False

Method is indirectly used by all core Bearer token issuing grant types:

- Authorization Code Grant
- Implicit Grant
- Resource Owner Password Credentials Grant
- Client Credentials Grant

class twitcher.oauth2.**CustomTokenValidator**(*secret*, *issuer*)

validate_bearer_token(*token*, *scopes*, *request*)

Ensure the Bearer token is valid and authorized access to scopes.

Parameters

- **token** – A string of random characters.
- **scopes** – A list of scopes associated with the protected resource.
- **request** – The HTTP Request (oauthlib.common.Request)

A key to OAuth 2 security and restricting impact of leaked tokens is the short expiration time of tokens, *always ensure the token has not expired!*.

Two different approaches to scope validation:

1) **all(scopes). The token must be authorized access to all scopes** associated with the resource. For example, the token has access to `read-only` and `images`, thus the client can view images but not upload new. Allows for fine grained access control through combining various scopes.

2) **any(scopes). The token must be authorized access to one of the** scopes associated with the resource. For example, token has access to `read-only-images`. Allows for fine grained, although arguably less convenient, access control.

A powerful way to use scopes would mimic UNIX ACLs and see a scope as a group with certain privileges. For a restful API these might map to HTTP verbs instead of read, write and execute.

Note, the request.user attribute can be set to the resource owner associated with this token. Similarly the request.client and request.scopes attribute can be set to associated client object and authorized scopes. If you then use a decorator such as the one provided for django these attributes will be made available in all protected views as keyword arguments.

Parameters

- **token** – Unicode Bearer token
- **scopes** – List of scopes (defined by you)
- **request** – The HTTP Request (oauthlib.common.Request)

Return type True or False

Method is indirectly used by all core Bearer token issuing grant types:

- Authorization Code Grant
- Implicit Grant
- Resource Owner Password Credentials Grant
- Client Credentials Grant

6.5 OWS Security

class twitcher.owssecurity.**OWSSecurity**

verify_request(*request*)

Verify that the service request is allowed.

This method verifies that the provided credentials are valid. Depending on the authentication configuration this could be a client X509 certificate or an OAuth2 token.

6.6 OWS Registry

class twitcher.owsregistry.**OWSRegistry**(*servicestore*)

OWS Service Registry is a service to register OWS services for the OWS proxy.

clear_services()

Removes all services from the service store.

get_service_by_name(*name*)

Gets service with given `name` from service store.

get_service_by_url(*url*)

Gets service with given `url` from service store.

list_services()

Lists all registered OWS services.

register_service(*name*, *url*, **args*, ***kwargs*)

Adds an OWS service with the given `name` and `url` to the service store.

unregister_service(*name*)

Removes OWS service with the given `name` from the service store.

6.7 OWS Proxy

The owsproxy is a service which acts as a proxy for registered OWS services.

The implementation of owsproxy is based on papyrus_ogcproxy

See also: https://github.com/nive/outpost/blob/master/outpost/proxy.py

CHAPTER

SEVEN

CHANGES

7.1 Unreleased

7.2 0.7.0 (2022-05-11)

Changes:

- Add request and response hooks operations to adapter allowing derived implementations to modify OWS proxied requests and returned responses from the service. The default adapter applies no modifications to the original definitions.

7.3 0.6.2 (2021-12-01)

Changes:

- Enforce regeneration of the `OWSRegistry` object on each request to avoid incorrect handling by adapters that require the new transaction or refreshed database session state each time.

7.4 0.6.1 (2021-10-27)

Changes:

- Integrate functionality changes of `0.5.x` branch back into `0.6.x`.
- Align `twitcher.adapter` features of `0.6.x` branch to support `0.5.x` behaviour.
- Revert removal of `ServiceStoreInterface` to provide relevant implementation details to external adapters.
- Apply missing interface classes as bases to default implementations.
- Add `owsproxy_uri` in frontpage response.
- Use `hybrid_property` to provide direct `twitcher.models.Service.verify` setter instead of protected `_verify`.
- Add more logging and handling of errors to catch cases where adapter doesn't return a valid `Service` instance.
- Add `scoped_session` to `session_factory` object to ensure distinct connections and transactions are created for concurrent requests.

7.5 0.6.0 (2020-04-01)

Changes:

- Added Keycloak support (#91).
- Added Keycloak demo notebook (#92, #93, #94).
- Refactor models definitions (`Service`, `Client`, `Token`).
- Refactor `twitcher.adapter` instantiation.
- Drop `rpcinterface` feature and endpoint.
- Drop `owsproxy_delegate` endpoint.
- Drop `owsproxy_secure` endpoint.

7.6 0.5.6 (2021-09-10)

Changes:

- Add Github Actions workflow to run local tests and Docker smoke tests for pre-validation of features and changes.
- Add Github issue, feature request and pull request templates.

Fixes:

- Pin packages `pyramid<2`, `zope.sqlalchemy>=1.5` and `sqlalchemy>=1.4,<2` to avoid errors with conflicting and upcoming release and features employed in code.
- Fix failing `cryptography` package build step in Docker image due to missing `g++` and `rust` dependencies (`rust` installed via `cargo`).

7.7 0.5.5 (2021-01-27)

Fixes:

- Update invalid reference to `python3-dev` in docker image. Travis-CI is also updated to run a smoke test build of this docker image prior to merge to help early detection of problems prior to deploy triggers from tags.

7.8 0.5.4 (2020-10-29)

Changes:

- Replace `waitress` by `gunicorn` to resolve issue related to slow download of large files (#97).

7.9 0.5.3 (2020-02-20)

Changes:

- Reduce log level of "`failed security check`" from `exception` to `warning` as it corresponds to the expected code behavior (unauthorised access) when `OWSException` is raised, instead of dumping an unhandled error traceback.

7.10 0.5.2 (2019-07-11)

New Features:

- Adds route `/info` which returns contents of `twitcher.__version__`.
- Adds route */versions* which returns version details such as *Twitcher* app version and employed adapter version.

Changes:

- Updated `README.rst` to match recent development, reference and docker image link.
- Adds URI of `/info` and `/versions` routes in the frontpage response.
- Corresponding HTTP status codes are returned for raised `OWSException`.

Fixes:

7.11 0.5.1 (2019-05-24)

New Features:

- Add *postgres* extra requirements for when it is used as database driver with `sqlalchemy`.

Changes:

- Use `container` instead of `config` for `AdapterInterface.owsproxy_config` to match real use cases.

Fixes:

- Improve the adapter import methodology to work with more use cases (Ouranosinc/Magpie#182).
- Fix incorrect setup for bump version within `Makefile`.
- Fix Twitcher `main` including `twitcher.<module>` instead of `.<module>`.

7.12 0.5.0 (2019-05-22)

Changes:

- Skipped Buildout (#49).
- Replaced mongodb by sqlalchemy (#51).
- Simplified `Makefile` and skipped conda targets (#75).
- Add `Makefile` targets for `docker`, `bumpversion` and `coverage` analysis related tasks (#67).
- Removed unused `config` module (#70).

New Features:

- Provided a `Dockerfile` for building *Twitcher* (#67).
- Provide `AdapterInterface` to allow overriding store implementations with configuration setting `twitcher.adapter` (#67).
- Add version auto-update (number and date) of these 'changes' with `bump2version` (#67).

Fixes:

- Update requirements with missing dependencies when building docker image.
- Various fixes (#71, #72, #73, #74)

7.13 0.4.0 (2019-05-02)

Changes:

- Skipped Python 2.7 support (#61).
- Added public URL "purl" (#58).
- Added SSL verify option (#55).
- Skipped internal WPS (#52).
- Moved tests to top-level folder (#47).

7.14 0.3.8 (2018-09-11)

Fixes:

- Fixed the wps DataInputs params encoding (#42).
- Fixed error 400 Contradictory scheme headers (#40).

New Features:

- make protected path configurable (#36).

7.15 0.3.7 (2018-03-13)

Fixes:

- Fixed exclude filter in `MANIFEST.in`.

New Features:

- Feature #28: use request upstream when not using WPS (e.g download file through `thredds`).

7.16 0.3.6 (2018-03-08)

- Fix PEP8
- Removed unused `c4i` option.
- Added `auth` option to set authentication method.
- Updated docs for usage of x509 certificates.

New Features:

- Feature #25: using x509 certificates for service authentication.

7.17 0.3.5 (2018-03-01)

- Fix PEP8.
- Updated makefile.
- Updated buildout recipes.
- Fixed nginx dependency.
- Updated mongodb 3.4.
- Configured csrf in `xmlrpc`.
- Fixed tutorial example.
- Added readthedocs, licence and chat badges.

7.18 0.3.4 (2017-05-05)

- Updated logging.
- Fixed: creates workdir if it does not exist.

7.19 0.3.3 (2017-04-27)

- Fixed fetching of access token when service is public.

7.20 0.3.2 (2017-01-31)

- Set header `X-X509-User-Proxy`.

7.21 0.3.1 (2017-01-26)

- Fix PEP8.
- Set permission of `certfile`.
- Added option `ows-proxy-delegate`.

7.22 0.3.0 (2017-01-11)

- Fix PEP8.
- Changed rpc interface.
- Added twitcher.client module.
- Using esgf scls service to get credentials.
- Updated internal pywps to version 4.0.0.
- Using default port 5000.
- Added ipython notebook examples.
- Moved `namesgenerator` to top-level.
- Added `_compat` module for Python 3.x/2.x compatibility.
- Added `twitcher.api` and cleaned up rpcinterface.
- Added `twitcher.store` with mongodb and memory implementation.
- Added `twitcher.datatype` with `AccessToken` and `Service`.
- Using https port only.
- Using `OWSExceptions` on errors in owsproxy.

7.23 0.2.4 (2016-12-23)

- Fix PEP8.
- Using `replace_caps_url` in `owsproxy`.
- Pinned `mongodb=2.6*|3.3.9`.
- Replaced `service_url` by `proxy_url`.
- Added `wms_130` and renamed `wms_111`.

7.24 0.2.3 (2016-11-18)

- Fix PEP8.
- Using `doc2dict`, renamed `get_service_by_name()`.
- Added support for c4i tokens.
- Updated deps: `pytest`, `mongodb`.
- Updated buildout recipes.
- Fixed functional tests.

7.25 0.2.2 (2016-08-18)

- Fix PEP8.
- Don't allow duplicate service names.

7.26 0.2.1 (2016-08-05)

- Register service with public access.
- WMS services can be registered.

7.27 0.2.0 (2016-07-18)

- Updated to new buildout with separated conda environment.
- Replaced nose by pytest.
- Updated installation docs.

7.28 0.1.7 (2016-06-09)

Fixes:

- Update of service failed (#17).

7.29 0.1.6 (2016-06-01)

- Updated docs.
- Renamed Python package to `pyramid_twitcher`.
- Conda `environment.yml` added.
- Using `get_sane_name()`.
- Replaced `httplib2` by `requests`.

Fixes:

- Don't check token for allowed requests (#14).
- Ignore decoding errors of response content (#13).
- Fixed twitcher app config: wrong egg name.

7.30 0.1.5 (2016-04-22)

- Fixed docs links

7.31 0.1.4 (2016-04-19)

- Fixed `MANIFEST.in`
- Fixed service database index.
- Updated `Makefile`.
- Added more links to appendix.

7.32 0.1.0 (2015-12-07)

Initial Release.

PYTHON MODULE INDEX

t

twitcher, 19
twitcher.oauth2, 22
twitcher.owsproxy, 25
twitcher.scripts.twitcherctl, 20

INDEX

A

add_client_app() (*twitcher.client.TwitcherService method*), 19

C

clear_services() (*twitcher.api.TwitcherAPI static method*), 21
clear_services() (*twitcher.client.TwitcherService method*), 19
clear_services() (*twitcher.owsregistry.OWSRegistry method*), 25
CustomTokenValidator (*class in twitcher.oauth2*), 23

F

fetch_token() (*twitcher.client.TwitcherService method*), 19

G

generate_token_view() (*in module twitcher.oauth2*), 22
get_service() (*twitcher.api.TwitcherAPI static method*), 21
get_service() (*twitcher.client.TwitcherService method*), 19
get_service_by_name() (*twitcher.owsregistry.OWSRegistry method*), 25
get_service_by_url() (*twitcher.owsregistry.OWSRegistry method*), 25

L

list_services() (*twitcher.api.TwitcherAPI static method*), 21
list_services() (*twitcher.client.TwitcherService method*), 19
list_services() (*twitcher.owsregistry.OWSRegistry method*), 25

M

module
    twitcher, 19
    twitcher.oauth2, 22
    twitcher.owsproxy, 25
    twitcher.scripts.twitcherctl, 20

O

OWSRegistry (*class in twitcher.owsregistry*), 25
OWSSecurity (*class in twitcher.owssecurity*), 25

R

RandomTokenValidator (*class in twitcher.oauth2*), 22
register_client_app_view() (*in module twitcher.oauth2*), 22
register_service() (*twitcher.api.TwitcherAPI static method*), 21
register_service() (*twitcher.client.TwitcherService method*), 19
register_service() (*twitcher.owsregistry.OWSRegistry method*), 25

S

save_bearer_token() (*twitcher.oauth2.RandomTokenValidator method*), 22
SignedTokenValidator (*class in twitcher.oauth2*), 23

T

twitcher
    module, 19
twitcher.oauth2
    module, 22
twitcher.owsproxy
    module, 25
twitcher.scripts.twitcherctl
    module, 20
TwitcherAPI (*class in twitcher.api*), 21
TwitcherCtl (*class in twitcher.scripts.twitcherctl*), 21
TwitcherService (*class in twitcher.client*), 19

U

unregister_service() (*twitcher.api.TwitcherAPI static method*), 21

unregister_service() (*twitcher.client.TwitcherService method*), 19
unregister_service() (*twitcher.owsregistry.OWSRegistry method*), 25

V

validate_bearer_token() (*twitcher.oauth2.CustomTokenValidator method*), 24
validate_bearer_token() (*twitcher.oauth2.RandomTokenValidator method*), 22
validate_bearer_token() (*twitcher.oauth2.SignedTokenValidator method*), 23
verify_request() (*twitcher.owssecurity.OWSSecurity method*), 25