Cliquet Documentation

Release 3.1.5

Mozilla Services â Da French Team

May 18, 2016

1 Rationale 3

2 Getting started 9

3 HTTP Protocol 13

4 Internals 37

5 Ecosystem 81

6 Troubleshooting 87

7 Contributing 89

8 Indices and tables 93

Python Module Index 95

Fig. 1: A cliquet, or ratchet, is a mechanical device that allows continuous linear or rotary motion in only one direction while preventing motion in the opposite direction.

A cliquet provides some basic but essential functionality — efficient in a variety of contexts, from bikes rear wheels to most advanced clockmaking!

CHAPTER 1

Rationale

Cliquet is a toolkit to ease the implementation of HTTP microservices. It is mainly focused on data-driven REST APIs (aka *CRUD*).

1.1 Philosophy

- *KISS*;
- No magic;
- Works with defaults;
- Easy customization;
- Straightforward component substitution.

Cliquet doesn't try to be a framework: any project built with *Cliquet* will expose a well defined HTTP protocol for:

- Collection and records manipulation;
- HTTP status and headers handling;
- API versioning and deprecation;
- Errors formatting.

This protocol is an implementation of a series of good practices (followed at Mozilla Services and elsewhere).

The goal is to produce standardized APIs, which follow some well known patterns, encouraging genericity in clients code [1].

Of course, *Cliquet* can be *extended* and customized in many ways. It can also be used in any kind of project, for its tooling, utilities and helpers.

1.2 Features

It is built around the notion of resources: resources are defined by sub-classing, and *Cliquet* brings up the HTTP endpoints automatically.

[1] Switch from custom protocol to JSON-API spec is being discussed.

1.2.1 Records and synchronization

- Collection of records by user;
- Optional validation from schema;
- Sorting and filtering;
- Pagination using continuation tokens;
- Polling for collection changes;
- Record race conditions handling using preconditions headers;
- Notifications channel (e.g. run asynchronous tasks on changes).

1.2.2 Generic endpoints

- Hello view at root url;
- Heartbeat for monitoring;
- Batch operations;
- API versioning and deprecation;
- Errors formatting;
- `Backoff` and `Retry-After` headers.

1.2.3 Toolkit

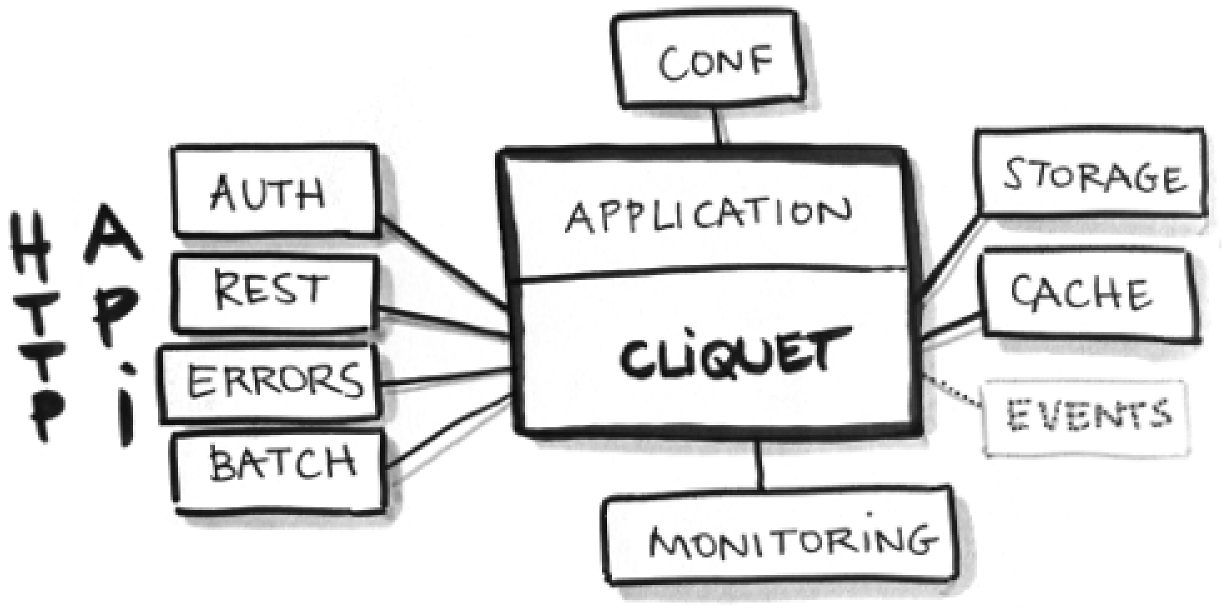

Fig. 1.1: *Cliquet* brings a set of simple but essential features to build APIs.

- Configuration through INI files or environment variables;
- Pluggable storage and cache backends;

- Pluggable authentication and user groups management;
- Pluggable authorization and permissions management;
- Structured logging;
- Monitoring tools;
- Profiling tools.

Pluggable components can be replaced by another one via configuration.

1.3 Dependencies

Cliquet is built on the shoulders of giants:

- Cornice for the REST helpers;
- Pyramid for the heavy HTTP stuff;
- SQLAlchemy core, for database sessions and pooling;

Everything else is meant to be **pluggable and optional**.

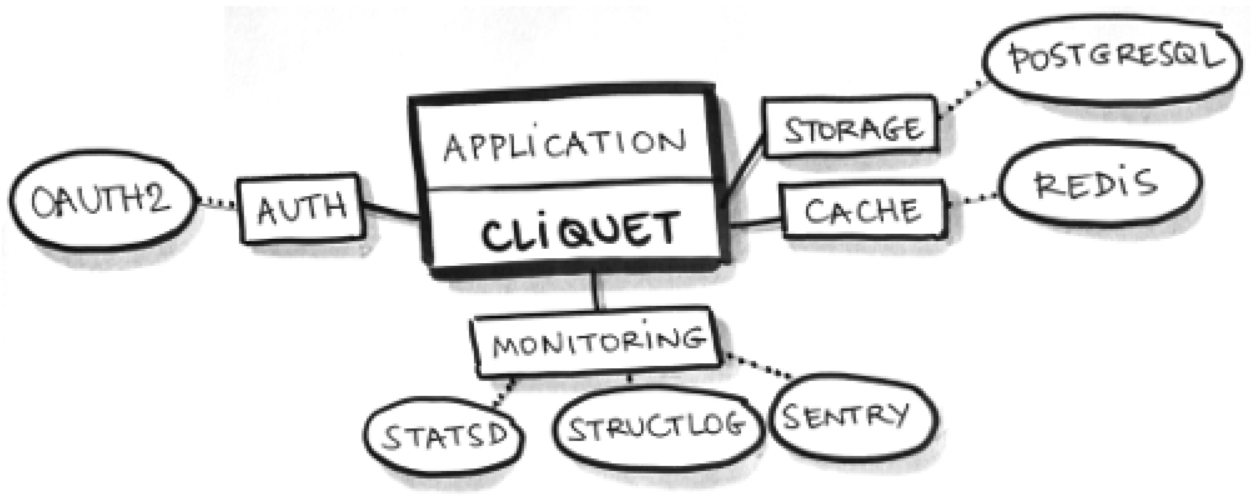

Fig. 1.2: Examples of configuration for a *Cliquet* application in production.

- *Basic Auth*, *FxA OAuth2* or any other source of authentication;
- *Default* or custom class for authorization logics;
- *PostgreSQL* for storage;
- *Redis* for key-value cache with expiration;
- *StatsD* metrics;
- *Sentry* reporting via logging;
- *NewRelic* database profiling (*for development*);
- *Werkzeug* Python code profiling (*for development*).

A *Cliquet* application can change or force default values for any setting.

1.4 Built with Cliquet

Some applications in the wild built with *Cliquet*:

- Reading List, a service to synchronize articles between devices;
- Kinto, a service to store and synchronize schema-less data.
- Syncto, a service to access *Firefox Sync* using *kinto.js*.
- *Please contact us to add yours.*

1.4.1 Context

(*to be done*)

- Cloud Services team at Mozilla
- ReadingList project story
- Firefox Sync
- Cloud storage
- Firefox OS User Data synchronization and backup

1.5 Vision

1.5.1 General

Any application built with *Cliquet*:

- follows the same conventions regarding the HTTP API;
- takes advantage of its component *pluggability*;
- can be *extended* using custom code or Pyramid external packages;

Let's build a *sane ecosystem* for microservices in Python!

1.5.2 Roadmap

The future features we plan to implement in *Cliquet* are currently driven by the use-cases we meet internally at Mozilla. Most notable are:

- Attachments on records (e.g. *Remote Storage* compatibility);
- Records generic indexing (e.g. streaming records to *ElasticSearch*).
- ... come and discuss enhancements in the issue tracker!

1.6 Similar projects

- Python Eve, built on Flask and MongoDB;
- *Please contact us to add more if any.*

Since the protocol is language independant and follows good HTTP/REST principles, in the long term *Cliquet* should become only one among several server implementations.

Note: We encourage you to implement a clone of this project — using Node.js, Asyncio, Go, Twisted, Django or anything else — following *the same protocol*!

CHAPTER 2

Getting started

2.1 Installation

```
$ pip install cliquet
```

More details about installation and storage backend is provided in *a dedicated section*.

2.2 Start a Pyramid project

As detailed in Pyramid documentation, create a minimal application, or use its scaffolding tool:

```
$ pcreate -s starter MyProject
```

2.2.1 Include Cliquet

In the application main file (e.g. `MyProject/myproject/__init__.py`), just add some extra initialization code:

```
import pkg_resources

import cliquet
from pyramid.config import Configurator

# Module version, as defined in PEP-0396.
__version__ = pkg_resources.get_distribution(__package__).version


def main(global_config, **settings):
    config = Configurator(settings=settings)

    cliquet.initialize(config, __version__)
    return config.make_wsgi_app()
```

By doing that, basic features like authentication, monitoring, error formatting, deprecation indicators are now available, and rely on configuration present in `development.ini`.

2.2.2 Run!

With some backends, like *PostgreSQL*, some tables and indices have to be created. A generic command is provided to accomplish this:

```
$ cliquet --ini development.ini migrate
```

Like any *Pyramid* application, it can be served locally with:

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

A *hello* view is now available at http://localhost:6543/v0/ (As well as basic endpoints like the *utilities*).

The next steps will consist in building a custom application using Cornice or **the Pyramid ecosystem**.

But most likely, it will consist in **defining REST resources** using *Cliquet* python API !

2.2.3 Authentication

Currently, if no *authentication is set in settings*, *Cliquet* relies on *Basic Auth*. It will associate a unique *user id* for every user/password combination.

Using HTTPie, it is as easy as:

```
$ http -v http://localhost:6543/v0/ --auth user:pass
```

Note: In the case of *Basic Auth*, there is no need of registering a user/password. Pick any combination, and include them in each request.

2.3 Define resources

In order to define a resource, inherit from `cliquet.resource.UserResource`, in a subclass, in `myproject/views.py` for example:

```python
from cliquet import resource

@resource.register()
class Mushroom(resource.UserResource):
    # No schema yet.
    pass
```

In application initialization, make *Pyramid* aware of it:

```python
def main(global_config, **settings):
    config = Configurator(settings=settings)

    cliquet.initialize(config, __version__)
    config.scan("myproject.views")
    return config.make_wsgi_app()
```

In order to bypass the installation and configuration of *Redis* or *PostgreSQL*, specify the «in-memory» backends in `development.ini`:

```
# development.ini
cliquet.cache_backend = cliquet.cache.memory
cliquet.storage_backend = cliquet.storage.memory
cliquet.permission_backend = cliquet.permission.memory
```

A Mushroom resource API is now available at the `/mushrooms/` URL.

It will accept a bunch of REST operations, as defined in the *API section*.

Warning: Without schema, a resource will not store any field at all!

The next step consists in attaching a schema to the resource, to control what fields are accepted and stored.

2.3.1 Schema validation

It is possible to validate records against a predefined schema, associated to the resource.

Currently, only Colander is supported, and it looks like this:

```
import colander
from cliquet import resource


class MushroomSchema(resource.ResourceSchema):
    name = colander.SchemaNode(colander.String())


@resource.register()
class Mushroom(resource.UserResource):
    mapping = MushroomSchema()
```

2.4 What's next ?

2.4.1 Configuration

See *Configuration* to customize the application settings, such as authentication, storage or cache backends.

2.4.2 Resource customization

See *the resource documentation* to specify custom URLs, schemaless resources, read-only fields, unicity constraints, record pre-processing...

2.4.3 Advanced initialization

`cliquet.`**`initialize`**(*config*, *version=None*, *project_name=''*, *default_settings=None*)

Initialize Cliquet with the given configuration, version and project name.

This will basically include cliquet in Pyramid and set route prefix based on the specified version.

Parameters

- **`config`** (*`Configurator`*) – Pyramid configuration

- **`version`** (*`str`*) – Current project version (e.g. '0.0.1') if not defined in application settings.
- **`project_name`** (*`str`*) – Project name if not defined in application settings.
- **`default_settings`** (*`dict`*) – Override cliquet default settings values.

2.4.4 Beyond Cliquet

Cliquet is just a component! The application can still be built and extended using the full *Pyramid* ecosystem.

See *the dedicated section* for examples of *Cliquet* extensions.

CHAPTER 3

HTTP Protocol

3.1 API Versioning

The *HTTP API* exposed by the service will be consumed by clients, like a Javascript client.

The *HTTP API* is subject to changes. It follows the *Cliquet Protocol*.

When the *HTTP API* is changed, its version is incremented. The *HTTP API* version follows a *Semantic Versioning* pattern and uses this rule to be incremented:

1. any change to the *HTTP API* that is backward compatible increments the **MINOR** number, and the modification in the documentation should reflect this with a header like “Added in 1.x”.
2. any change to the *HTTP API* that is backward incompatible increments the **MAJOR** number, and the differences are summarized at the begining of the documentation, a new document for that **MAJOR** version is created.

Note: We’re not using the **PATCH** level of Semantic Versioning, since bug fixes have no impact on the exposed HTTP API; if they do MINOR or MAJOR should be incremented.

We want to avoid **MAJOR** changes as much as possible in the future, and stick with 1.x as long as we can.

A client that interacts with the service can query the server to know what is its *HTTP API* version. This is done with a query on the root view, as described in *the root API description*.

If a client relies on a feature that was introduced at a particular version, it should check that the server implements the minimal required version.

The JSON response body contains an **http_api_version** key which value is the **MAJOR.MINOR** version.

3.2 Authentication

Depending on the authentication policies initialized in the application, the HTTP method to authenticate requests may differ.

A policy based on *OAuth2 bearer tokens* is recommended, but not mandatory. See *configuration* for further information.

In the current implementation, when multiple policies are configured, *user identifiers* are isolated by policy. In other words, there is no way to access the same set of records using different authentication methods.

By default, a relatively secure *Basic Auth* is enabled.

3.2.1 Basic Auth

If enabled in configuration, using a *Basic Auth* token will associate a unique *user identifier* to an username/password combination.

```
Authorization: Basic <basic_token>
```

The token shall be built using this formula `base64("username:password")`.

Empty passwords are accepted, and usernames can be anything (custom, UUID, etc.)

If the token has an invalid format, or if *Basic Auth* is not enabled, this will result in a `401` error response.

Warning: Since *user id* is derived from username and password, there is no way to change the password without loosing access to existing records.

3.2.2 OAuth Bearer token

If the configured authentication policy uses *OAuth2 bearer tokens*, authentication shall be done using this header:

```
Authorization: Bearer <oauth_token>
```

The policy will verify the provided *OAuth2 bearer token* on a remote server.

notes If the token is not valid, this will result in a `401` error response.

3.2.3 Firefox Accounts

In order to enable authentication with *Firefox Accounts*, install and configure mozilla-services/cliquet-fxa.

3.3 Resource endpoints

All *endpoints* URLs are prefixed by the major version of the *HTTP API* (e.g /v1 for 1.4).

e.g. the URL for all the endpoints is structured as follows::

```
https://<server name>/<api MAJOR version>/<further instruction>
```

The full URL prefix will be implied throughout the rest of this document and it will only describe the **<further instruction>** part.

3.3.1 GET /{collection}

Requires authentication

Returns all records of the current user for this collection.

The returned value is a JSON mapping containing:

- `data`: the list of records, with exhaustive fields;

A `Total-Records` response header indicates the total number of records of the collection.

A `Last-Modified` response header provides a human-readable (rounded to second) of the current collection timestamp.

For cache and concurrency control, an `ETag` response header gives the value that consumers can provide in subsequent requests using `If-Match` and `If-None-Match` headers (see *section about timestamps*).

Request:

```
GET /articles HTTP/1.1
Accept: application/json
Authorization: Basic bWF0Og==
Host: localhost:8000
```

Response:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Access-Control-Expose-Headers: Backoff, Retry-After, Alert, Content-Length, ETag, Next-Page, Total-Re
Content-Length: 436
Content-Type: application/json; charset=UTF-8
Date: Tue, 28 Apr 2015 12:08:11 GMT
Last-Modified: Mon, 12 Apr 2015 11:12:07 GMT
ETag: "1430222877724"
Total-Records: 2

{
    "data": [
        {
            "id": "dc86afa9-a839-4ce1-ae02-3d538b75496f",
            "last_modified": 1430222877724,
            "title": "MoCo",
            "url": "https://mozilla.com",
        },
        {
            "id": "23160c47-27a5-41f6-9164-21d46141804d",
            "last_modified": 1430140411480,
            "title": "MoFo",
            "url": "https://mozilla.org",
        }
    ]
}
```

Filtering

Single value

- `/collection?field=value`

Minimum and maximum

Prefix field name with `min_` or `max_`:

- `/collection?min_field=4000`

Note: The lower and upper bounds are inclusive (*i.e equivalent to greater or equal*).

Note: `lt_` and `gt_` can also be used to exclude the bound.

Multiple values

Prefix field with `in_` and provide comma-separated values.

- `/collection?in_status=1,2,3`

Exclude

Prefix field name with `not_`:

- `/collection?not_field=0`

Exclude multiple values

Prefix field name with `exclude_`:

- `/collection?exclude_field=0,1`

Note: Will return an error if a field is unknown.

Note: The `ETag` and `Last-Modified` response headers will always be the same as the unfiltered collection.

Sorting

- `/collection?_sort=-last_modified,field`

Note: Ordering on a boolean field gives `true` values first.

Note: Will return an error if a field is unknown.

Counting

In order to count the number of records, for a specific field value for example, without fetching the actual collection, a `HEAD` request can be used. The `Total-Records` response header will then provide the total number of records.

See *batch endpoint* to count several collections in one request.

Polling for changes

The `_since` parameter is provided as an alias for `gt_last_modified`.

- `/collection?_since=1437035923844`

When filtering on `last_modified` every deleted records will appear in the list with a `deleted` flag and a `last_modified` value that corresponds to the deletion event.

If the `If-None-Match: "<timestamp>"` request header is provided as described in the *section about timestamps* and if the collection was not changed, a `304 Not Modified` response is returned.

Note: The `_before` parameter is also available, and is an alias for `lt_last_modified` (*strictly inferior*).

Note: `_since` and `_before` also accept a value between quotes (`"`) as it would be returned in the `ETag` response header (see *response timestamps*).

Request:

```
GET /articles?_since=1437035923844 HTTP/1.1
Accept: application/json
Authorization: Basic bWF0Og==
Host: localhost:8000
```

Response:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Access-Control-Expose-Headers: Backoff, Retry-After, Alert, Content-Length, ETag, Next-Page, Total-Re
Content-Length: 436
Content-Type: application/json; charset=UTF-8
Date: Tue, 28 Apr 2015 12:08:11 GMT
Last-Modified: Mon, 12 Apr 2015 11:12:07 GMT
ETag: "1430222877724"
Total-Records: 2

{
    "data": [
        {
            "id": "dc86afa9-a839-4ce1-ae02-3d538b75496f",
            "last_modified": 1430222877724,
            "title": "MoCo",
            "url": "https://mozilla.com",
        },
        {
            "id": "23160c47-27a5-41f6-9164-21d46141804d",
            "last_modified": 1430140411480,
            "title": "MoFo",
            "url": "https://mozilla.org",
        },
        {
            "id": "11130c47-37a5-41f6-9112-32d46141804f",
            "deleted": true,
            "last_modified": 1430140411480
        }
    ]
}
```

Paginate

If the `_limit` parameter is provided, the number of records returned is limited.

If there are more records for this collection than the limit, the response will provide a `Next-Page` header with the URL for the Next-Page.

When there is no more `Next-Page` response header, there is nothing more to fetch.

Pagination works with sorting, filtering and polling.

Note: The `Next-Page` URL will contain a continuation token (`_token`).

It is recommended to add precondition headers (`If-Match` or `If-None-Match`), in order to detect changes on collection while iterating through the pages.

Partial response

If the `_fields` parameter is provided, only the fields specified are returned. Fields are separated with a comma.

This is vital in mobile contexts where bandwidth usage must be optimized.

Nested objects fields are specified using dots (e.g. `address.street`).

Note: The `id` and `last_modified` fields are always returned.

Request:

```
GET /articles?_fields=title,url
Accept: application/json
Authorization: Basic bWF0Og==
Host: localhost:8000
```

Response:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Access-Control-Expose-Headers: Backoff, Retry-After, Alert, Content-Length, ETag, Next-Page, Total-Re
Content-Length: 436
Content-Type: application/json; charset=UTF-8
Date: Tue, 28 Apr 2015 12:08:11 GMT
Last-Modified: Mon, 12 Apr 2015 11:12:07 GMT
ETag: "1430222877724"
Total-Records: 2

{
    "data": [
        {
            "id": "dc86afa9-a839-4ce1-ae02-3d538b75496f",
            "last_modified": 1430222877724,
            "title": "MoCo",
            "url": "https://mozilla.com",
        },
        {
            "id": "23160c47-27a5-41f6-9164-21d46141804d",
            "last_modified": 1430140411480,
            "title": "MoFo",
            "url": "https://mozilla.org",
        }
    ]
}
```

List of available URL parameters

- `<prefix?><field name>`: filter by value(s)
- `_since`, `_before`: polling changes
- `_sort`: order list

- `_limit`: pagination max size
- `_token`: pagination token
- `_fields`: filter the fields of the records

Filtering, sorting, partial responses and paginating can all be combined together.

- `/collection?_sort=-last_modified&_limit=100&_fields=title`

HTTP Status Codes

- `200 OK`: The request was processed
- `304 Not Modified`: Collection did not change since value in `If-None-Match` header
- `400 Bad Request`: The request querystring is invalid
- `401 Unauthorized`: The request is missing authentication headers
- `403 Forbidden`: The user is not allowed to perform the operation, or the resource is not accessible
- `406 Not Acceptable`: The client doesn't accept supported responses Content-Type
- `412 Precondition Failed`: Collection changed since value in `If-Match` header

3.3.2 POST /{collection}

Requires authentication

Used to create a record in the collection. The POST body is a JSON mapping containing:

- `data`: the values of the resource schema fields;
- `permissions`: *optional* a json dict containing the permissions for the record to be created.

The POST response body is a JSON mapping containing:

- `data`: the newly created record, if all posted values are valid;
- `permissions`: *optional* a json dict containing the permissions for the requested resource.

If the `If-Match: "<timestamp>"` request header is provided as described in the *section about timestamps*, and if the collection has changed meanwhile, a `412 Precondition failed` error is returned.

If the `If-None-Match: *` request header is provided, and if the provided `data` contains an `id` field, and if there is already an existing record with this `id`, a `412 Precondition failed` error is returned.

Request:

```
POST /articles HTTP/1.1
Accept: application/json
Authorization: Basic bWF0Og==
Content-Type: application/json; charset=utf-8
Host: localhost:8000

{
    "data": {
        "title": "Wikipedia FR",
        "url": "http://fr.wikipedia.org"
    }
}
```

Response:

```
HTTP/1.1 201 Created
Access-Control-Allow-Origin: *
Access-Control-Expose-Headers: Backoff, Retry-After, Alert, Content-Length
Content-Length: 422
Content-Type: application/json; charset=UTF-8
Date: Tue, 28 Apr 2015 12:35:02 GMT

{
    "data": {
        "id": "cd30c031-c208-4fb9-ad65-1582d2a7ad5e",
        "last_modified": 1430224502529,
        "title": "Wikipedia FR",
        "url": "http://fr.wikipedia.org"
    }
}
```

Validation

If the posted values are invalid (e.g. *field value is not an integer*) an error response is returned with status `400`.

See *details on error responses*.

Conflicts

Since some fields can be defined as unique per collection, some conflicts may appear when creating records.

Note: Empty values are not taken into account for field unicity.

Note: Deleted records are not taken into account for field unicity.

If a conflict occurs, an error response is returned with status `409`. A `details` attribute in the response provides the offending record and field name. See *dedicated section about errors*.

Timestamp

When a record is created, the timestamp of the collection is incremented.

It is possible to force the timestamp if the specified record has a `last_modified` field.

If the specified timestamp is in the past, the collection timestamp does not take the value of the created record but is bumped into the future as usual.

HTTP Status Codes

- `200 OK`: This record already exists, the one stored on the database is returned
- `201 Created`: The record was created
- `400 Bad Request`: The request body is invalid
- `401 Unauthorized`: The request is missing authentication headers

- `403 Forbidden`: The user is not allowed to perform the operation, or the resource is not accessible
- `406 Not Acceptable`: The client doesn't accept supported responses Content-Type
- `409 Conflict`: Unicity constraint on fields is violated
- `412 Precondition Failed`: Collection changed since value in `If-Match` header
- `415 Unsupported Media Type`: The client request was not sent with a correct Content-Type

3.3.3 DELETE /{collection}

Requires authentication

Delete multiple records. **Disabled by default**, see *Configuration*.

The DELETE response is a JSON mapping containing:

- `data`: list of records that were deleted, without schema fields.

It supports the same filtering capabilities as GET.

If the `If-Match: "<timestamp>"` request header is provided, and if the collection has changed meanwhile, a `412 Precondition failed` error is returned.

Request:

```
DELETE /articles HTTP/1.1
Accept: application/json
Authorization: Basic bWF0Og==
Host: localhost:8000
```

Response:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Access-Control-Expose-Headers: Backoff, Retry-After, Alert, Content-Length
Content-Length: 193
Content-Type: application/json; charset=UTF-8
Date: Tue, 28 Apr 2015 12:38:36 GMT

{
    "data": [
        {
            "deleted": true,
            "id": "cd30c031-c208-4fb9-ad65-1582d2a7ad5e",
            "last_modified": 1430224716097
        },
        {
            "deleted": true,
            "id": "dc86afa9-a839-4ce1-ae02-3d538b75496f",
            "last_modified": 1430224716098
        }
    ]
}
```

HTTP Status Codes

- `200 OK`: The records were deleted
- `401 Unauthorized`: The request is missing authentication headers

- `403 Forbidden`: The user is not allowed to perform the operation, or the resource is not accessible
- `405 Method Not Allowed`: This endpoint is not available
- `406 Not Acceptable`: The client doesn't accept supported responses Content-Type
- `412 Precondition Failed`: Collection changed since value in `If-Match` header

3.3.4 GET /{collection}/<id>

Requires authentication

Returns a specific record by its id. The GET response body is a JSON mapping containing:

- `data`: the record with exhaustive schema fields;
- `permissions`: *optional* a json dict containing the permissions for the requested record.

If the `If-None-Match: "<timestamp>"` request header is provided, and if the record has not changed meanwhile, a `304 Not Modified` is returned.

Request:

```
GET /articles/d10405bf-8161-46a1-ac93-a1893d160e62 HTTP/1.1
Accept: application/json
Authorization: Basic bWF0Og==
Host: localhost:8000
```

Response:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Access-Control-Expose-Headers: Backoff, Retry-After, Alert, Content-Length, ETag, Last-Modified
Content-Length: 438
Content-Type: application/json; charset=UTF-8
Date: Tue, 28 Apr 2015 12:42:42 GMT
ETag: "1430224945242"

{
    "data": {
        "id": "d10405bf-8161-46a1-ac93-a1893d160e62",
        "last_modified": 1430224945242,
        "title": "No backend",
        "url": "http://nobackend.org"
    }
}
```

HTTP Status Code

- `200 OK`: The request was processed
- `304 Not Modified`: Record did not change since value in `If-None-Match` header
- `401 Unauthorized`: The request is missing authentication headers
- `403 Forbidden`: The user is not allowed to perform the operation, or the resource is not accessible
- `406 Not Acceptable`: The client doesn't accept supported responses Content-Type
- `412 Precondition Failed`: Record changed since value in `If-Match` header

3.3.5 DELETE /{collection}/<id>

Requires authentication

Delete a specific record by its id.

The DELETE response is the record that was deleted. The DELETE response is a JSON mapping containing:

- `data`: the record that was deleted, without schema fields.

If the record is missing (or already deleted), a `404 Not Found` is returned. The consumer might decide to ignore it.

If the `If-Match` request header is provided, and if the record has changed meanwhile, a `412 Precondition failed` error is returned.

Note: Once deleted, a record will appear in the collection when polling for changes, with a deleted status (`delete=true`) and will have most of its fields empty.

Timestamp

When a record is deleted, the timestamp of the collection is incremented.

It is possible to force the timestamp by passing it in the querystring with `?last_modified=<value>`.

If the specified timestamp is in the past, the collection timestamp does not take the value of the deleted record but is bumped into the future as usual.

HTTP Status Code

- `200 OK`: The record was deleted
- `401 Unauthorized`: The request is missing authentication headers
- `403 Forbidden`: The user is not allowed to perform the operation, or the resource is not accessible
- `406 Not Acceptable`: The client doesn't accept supported responses Content-Type.
- `412 Precondition Failed`: Record changed since value in `If-Match` header

3.3.6 PUT /{collection}/<id>

Requires authentication

Create or replace a record with its id. The PUT body is a JSON mapping containing:

- `data`: the values of the resource schema fields;
- `permissions`: *optional* a json dict containing the permissions for the record to be created/replaced.

The PUT response body is a JSON mapping containing:

- `data`: the newly created/updated record, if all posted values are valid;
- `permissions`: *optional* the newly created permissions dict, containing the permissions for the created record.

Validation and conflicts behaviour is similar to creating records (`POST`).

If the `If-Match:  "<timestamp>"` request header is provided as described in the *section about timestamps*, and if the record has changed meanwhile, a `412 Precondition failed` error is returned.

If the `If-None-Match:  *` request header is provided and if there is already an existing record with this `id`, a `412 Precondition failed` error is returned.

Request:

```
PUT /articles/d10405bf-8161-46a1-ac93-a1893d160e62 HTTP/1.1
Accept: application/json
Authorization: Basic bWF0Og==
Content-Type: application/json; charset=utf-8
Host: localhost:8000

{
    "data": {
        "title": "Static apps",
        "url": "http://www.staticapps.org"
    }
}
```

Response:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Access-Control-Expose-Headers: Backoff, Retry-After, Alert, Content-Length
Content-Length: 439
Content-Type: application/json; charset=UTF-8
Date: Tue, 28 Apr 2015 12:46:36 GMT
ETag: "1430225196396"

{
    "data": {
        "id": "d10405bf-8161-46a1-ac93-a1893d160e62",
        "last_modified": 1430225196396,
        "title": "Static apps",
        "url": "http://www.staticapps.org"
    }
}
```

Timestamp

When a record is created or replaced, the timestamp of the collection is incremented.

It is possible to force the timestamp if the specified record has a `last_modified` field.

For replace, if the specified timestamp is less or equal than the existing record, the value is simply ignored and the timestamp is bumped into the future as usual.

For creation, if the specified timestamp is in the past, the collection timestamp does not take the value of the created/updated record but is bumped into the future as usual.

HTTP Status Code

- `201 Created`: The record was created
- `200 OK`: The record was replaced

- `400 Bad Request`: The record is invalid
- `401 Unauthorized`: The request is missing authentication headers
- `403 Forbidden`: The user is not allowed to perform the operation, or the resource is not accessible
- `406 Not Acceptable`: The client doesn't accept supported responses Content-Type.
- `409 Conflict`: If replacing this record violates a field unicity constraint
- `412 Precondition Failed`: Record was changed or deleted since value in `If-Match` header.
- `415 Unsupported Media Type`: The client request was not sent with a correct Content-Type.

3.3.7 PATCH /{collection}/<id>

Requires authentication

Modify a specific record by its id. The PATCH body is a JSON mapping containing:

- `data`: a subset of the resource schema fields (*key-value replace*);
- `permissions`: *optional* a json dict containing the permissions for the record to be modified.

The PATCH response body is a JSON mapping containing:

- `data`: the modified record (*full by default*);
- `permissions`: *optional* the modified permissions dict, containing the permissions for the modified record.

If a `Response-Behavior` request header is set to `light`, only the fields whose value was changed are returned. If set to `diff`, only the fields whose value became different than the one provided are returned.

Request:

```
PATCH /articles/d10405bf-8161-46a1-ac93-a1893d160e62 HTTP/1.1
Accept: application/json
Authorization: Basic bWF0Og==
Content-Type: application/json; charset=utf-8
Host: localhost:8000

{
    "data": {
        "title": "No Backend"
    }
}
```

Response:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Access-Control-Expose-Headers: Backoff, Retry-After, Alert, Content-Length
Content-Length: 439
Content-Type: application/json; charset=UTF-8
Date: Tue, 28 Apr 2015 12:46:36 GMT
ETag: "1430225196396"

{
    "data": {
        "id": "d10405bf-8161-46a1-ac93-a1893d160e62",
        "last_modified": 1430225196396,
        "title": "No Backend",
        "url": "http://nobackend.org"
    }
}
```

If the record is missing (or already deleted), a `404 Not Found` error is returned. The consumer might decide to ignore it.

If the `If-Match: "<timestamp>"` request header is provided as described in the *section about timestamps*, and if the record has changed meanwhile, a `412 Precondition failed` error is returned.

Note: `last_modified` is updated to the current server timestamp, only if a field value was changed.

Note: JSON-Patch is currently not supported. Any help is welcomed though!

Read-only fields

If a read-only field is modified, a `400 Bad request` error is returned.

Conflicts

If changing a record field violates a field unicity constraint, a `409 Conflict` error response is returned (see *error channel*).

Timestamp

When a record is modified, the timestamp of the collection is incremented.

It is possible to force the timestamp if the specified record has a `last_modified` field.

If the specified timestamp is less or equal than the existing record, the value is simply ignored and the timestamp is bumped into the future as usual.

HTTP Status Code

- `200 OK`: The record was modified
- `400 Bad Request`: The request body is invalid, or a read-only field was modified
- `401 Unauthorized`: The request is missing authentication headers
- `403 Forbidden`: The user is not allowed to perform the operation, or the resource is not accessible
- `406 Not Acceptable`: The client doesn't accept supported responses Content-Type.
- `409 Conflict`: If modifying this record violates a field unicity constraint
- `412 Precondition Failed`: Record changed since value in `If-Match` header
- `415 Unsupported Media Type`: The client request was not sent with a correct Content-Type.

3.3.8 Notes on permissions attribute

Shareable resources allow *permissions* management via the `permissions` attribute in the JSON payloads, along the `data` attribute. Permissions can be replaced or modified independently from data.

On a request, `permissions` is a JSON dict with the following structure:

```
"permissions": {<permission>: [<list_of_principals>]}
```

Where `<permission>` is the permission name (e.g. `read`, `write`) and `<list_of_principals>` should be replaced by an actual list of *principals*.

Example:

```
{
    "data": {
        "title": "No Backend"
    },
    "permissions": {
        "write": ["twitter:leplatrem", "group:ldap:42"],
        "read": ["system.Authenticated"]
    }
}
```

In a response, `permissions` contains the current permissions of the record (i.e. the *modified* version in case of a creation/modification).

Note: When a record is created or modified, the current *user id* **is always added** among the `write` principals.

Read more about leveraging resource permissions.

3.4 Batch operations

3.4.1 POST /batch

Requires authentication

The POST body is a mapping, with the following attributes:

- `requests`: the list of requests
- `defaults`: (*optional*) default requests values in common for all requests

  Each request is a JSON mapping, with the following attribute:

- `method`: HTTP verb
- `path`: URI
- `body`: a mapping
- `headers`: (*optional*), otherwise take those of batch request

```
POST /batch HTTP/1.1
Accept: application/json
Accept-Encoding: gzip, deflate
Content-Length: 728
Host: localhost:8888
```

```
User-Agent: HTTPie/0.9.2

{
  "defaults": {
    "method" : "POST",
    "path" : "/articles",
  },
  "requests": [
    {
      "body" : {
        "data" : {
          "title": "MoFo",
          "url" : "http://mozilla.org",
          "added_by": "FxOS",
        },
        "permissions": {
          "read": ["system.Everyone"]
        }
      }
    },
    {
      "body" : {
        "data" : {
          "title": "MoCo",
          "url" : "http://mozilla.com"
          "added_by": "FxOS",
        }
      }
    },
    {
      "method" : "PATCH",
      "path" : "/articles/409",
      "body" : {
        "data" : {
          "read_position" : 3477
        }
      }
      "headers" : {
        "Response-Behavior": "light"
      }
    }
  ]
}
```

The response body is a list of all responses:

```
HTTP/1.1 200 OK
Access-Control-Expose-Headers: Retry-After, Content-Length, Alert, Backoff
Content-Length: 1674
Date: Wed, 17 Feb 2016 18:44:39 GMT
Server: waitress

{
  "responses": [
    {
      "status": 201,
      "path" : "/articles",
      "body" : {
        "data" : {
          "id": 411,
          "title": "MoFo",
          "url" : "http://mozilla.org",
          ...
        }
      },
      "headers": {
        ...
      }
    },
    {
      "status": 201,
      "path" : "/articles",
      "body" : {
        "data" : {
          "id": 412,
          "title": "MoCo",
          "url" : "http://mozilla.com",
          ...
        }
      },
      "headers": {
        ...
      }
    },
    {
      "status": 200,
      "path" : "/articles/409",
      "body" : {
        "data" : {
          "id": 409,
          "url": "...",
          ...
          "read_position" : 3477
        }
      },
      "headers": {
        ...
      }
    }
  ]
}
```

HTTP Status Codes

- `200 OK`: The request has been processed
- `400 Bad Request`: The request body is invalid
- `50X`: One of the sub-request has failed with a `50X` status

Warning: Since the requests bodies are necessarily mappings, posting arbitrary data (*like raw text or binary*) is not supported.

Note: Responses are executed and provided in the same order than requests.

About transactions

The whole batch of requests is executed under one transaction only.

In order words, if one of the sub-request fails with a 503 status for example, then every previous operation is rolled back.

Important: With the current implementation, if a sub-request fails with a 4XX status (eg. `412 Precondition failed` or `403 Unauthorized` for example) the transaction is **not** rolled back.

3.5 Utility endpoints for OPS and Devs

3.5.1 GET /

The returned value is a JSON mapping containing:

Changed in version 2.12.

- `project_name`: the name of the service (e.g. `"reading list"`)
- `project_docs`: The URL to the service documentation. (this document!)
- `project_version`: complete application/project version (`"3.14.116"`)
- `http_api_version`: the MAJOR.MINOR version of the exposed HTTP API (`"1.1"`) defined in configuration.
- `cliquet_protocol_version`: the cliquet protocol version (`"2"`)
- `url`: absolute URI (without a trailing slash) of the API (*can be used by client to build URIs*)
- `eos`: date of end of support in ISO 8601 format (`"yyyy-mm-dd"`, undefined if unknown)
- `settings`: a mapping with the values of relevant public settings for clients
  - `batch_max_requests`: Number of requests that can be made in a batch request.
  - `readonly`: Only requests with read operations are allowed.
- `capabilities`: a mapping used by clients to detect optional features of the API.
  - Example:

```
{
  "auth-fxa": {
    "description": "Firefox Account authentication",
    "url": "http://github.com/mozilla-services/cliquet-fxa"
  }
}
```

Optional

- `user`: A mapping with an `id` field for the currently connected user id. The field is not present when no Authorization header is provided.

Note: The `project_version` contains the source code version, whereas the `http_api_version` contains the exposed *HTTP API* version.

The source code of the service can suffer changes and have its *project version* incremented, without impacting the publicly exposed HTTP API.

The `cliquet_protocol_version` is an internal notion tracking the version for some aspects of the API (e.g. synchronization of REST resources, utilities endpoints, etc.). It will differ from the `http_api_version` since the service will provide additionnal endpoints and conventions.

3.5.2 GET /__heartbeat__

Return the status of each service the application depends on. The returned value is a JSON mapping containing:

- `storage` true if storage backend is operational
- `cache` true if cache backend operational
- `permission` true if permission backend operational

If `cliquet-fxa` is installed, an additional key is present:

- `oauth` true if authentication is operational

Return `200` if the connection with each service is working properly and `503` if something doesn't work.

3.5.3 GET /__lbheartbeat__

Always return `200` with empty body.

Unlike the `__heartbeat__` health check endpoint, which return an error when backends and other upstream services are unavailable, this should always return 200.

This endpoint is suitable for a load balancer membership test. It the load balancer cannot obtain a response from this endpoint, it will stop sending traffic to the instance and replace it.

3.6 Server timestamps

In order to avoid race conditions, each change is guaranteed to increment the timestamp of the related collection. If two changes happen at the same millisecond, they will still have two different timestamps.

The `ETag` header with the current timestamp of the collection for the current user will be given on collection endpoints.

```
ETag: "1432208041618"
```

On record enpoints, the `ETag` header value will contain the timestamp of the record.

In order to bypass costly and error-prone HTTP date parsing, `ETag` headers are not HTTP date values.

A human readable version of the timestamp (rounded to second) is provided though in the `Last-Modified` response headers:

```
Last-Modified: Wed May 20 17:22:38 2015 +0200
```

Changed in version 2.0: In previous versions, cache and concurrency control was handled using `If-Modified-Since` and `If-Unmodified-Since`. But since the HTTP date does not include milliseconds, they contained the milliseconds timestamp as integer. The current version using `ETag` is HTTP compliant (see original discussion.)

Note: The client may send `If-Unmodified-Since` or `If-Modified-Since` requests headers, but in the current implementation, they will be ignored.

Important: When collection is empty, its timestamp remains the same until new records are created.

3.6.1 Cache control

In order to check that the client version has not changed, a `If-None-Match` request header can be used. If the response is `304 Not Modified` then the cached version is still good.

	GET
If-None-Match: "<timestamp>"	
Changed meanwhile	Return response content
Not changed	Empty `HTTP 304`

3.6.2 Concurrency control

In order to prevent race conditions, like overwriting changes occured in the interim for example, a `If-Match: "timestamp"` request header can be used. If the response is `412 Precondition failed` then the resource has changed meanwhile.

Concurrency control also allows to make sure a creation won't overwrite any record using the `If-None-Match: *` request header.

The following table gives a summary of the expected behaviour of a resource:

	POST	PUT	PATCH	DELETE
If-Match: "timestamp"				
Changed meanwhile	`HTTP 412`	`HTTP 412`	`HTTP 412`	`HTTP 412`
Not changed	Create	Overwrite	Modify	Delete
If-None-Match: *				
Id exists	`HTTP 412`	`HTTP 412`	No effect	No effect
Id unknown	Create	Create	No effect	No effect

When the client receives a `412 Precondition failed`, it can then choose to:

- overwrite by repeating the request without concurrency control;
- reconcile the resource by fetching, merging and repeating the request.

3.6.3 Replication

In order to replicate the timestamps when importing existing records, it is possible to force the last modified values.

When a record is created (via POST or PUT), the specified timestamp becomes the new collection timestamp if it is in the future (i.e. greater than current one). If it is in the past, the record is created with the timestamp in the past but the collection timestamp is bumped into the future as usual.

When a record is replaced, modified or deleted, if the specified timestamp is less or equal than the existing record, the value is simply ignored and the timestamp is bumped into the future as usual.

See *the resource endpoints documentation*.

3.7 Backoff indicators

3.7.1 Backoff header on heavy load

A `Backoff` header will be added to the success responses (>=200 and <400) when the server is under heavy load. It provides the client with a number of seconds during which it should avoid doing unnecessary requests.

```
Backoff: 30
```

Note: The back-off time is configurable on the server.

Note: In other implementations at Mozilla, there was `X-Weave-Backoff` and `X-Backoff` but the `X-` prefix for header has been deprecated since.

3.7.2 Retry-After indicators

A `Retry-After` header will be added if response is an error (>=500). See more details about *error responses*.

3.8 Error responses

3.8.1 Protocol description

Every response is JSON.

If the HTTP status is not OK (<200 or >=400), the response contains a JSON mapping, with the following attributes:

- `code`: matches the HTTP status code (e.g `400`)
- `errno`: stable application-level error number (e.g. `109`)
- `error`: string description of error type (e.g. `"Bad request"`)
- `message`: context information (e.g. `"Invalid request parameters"`)
- `info`: online resource (e.g. URL to error details)
- `details`: additional details (e.g. list of validation errors)

Example response

```
{
    "code": 412,
    "errno": 114,
    "error": "Precondition Failed",
    "message": "Resource was modified meanwhile",
    "info": "https://server/docs/api.html#errors",
}
```

Refer yourself to the ref:*set of errors codes <errors>*.

3.8.2 Retry-After indicators

A `Retry-After` header will be added to error responses (>=500), telling the client how many seconds it should wait before trying again.

```
Retry-After: 30
```

3.8.3 Precondition errors

As detailed in the *timestamps* section, it is possible to add concurrency control using `ETag` request headers.

When a concurrency error occurs, a `412 Precondition Failed` error response is returned.

Additional information about the record currently stored on the server will be provided in the `details` field:

```
{
    "code": 412,
    "errno": 114,
    "error":"Precondition Failed"
    "message": "Resource was modified meanwhile",
    "details": {
        "existing": {
            "last_modified": 1436434441550,
            "id": "00dd028f-16f7-4755-ab0d-e0dc0cb5da92",
            "title": "Original title"
        }
    },
}
```

3.8.4 Conflict errors

When a record violates unicity constraints, a `409 Conflict` error response is returned.

Additional information about conflicting record and field name will be provided in the `details` field.

```
{
    "code": 409,
    "errno": 122,
    "error": "Conflict",
    "message": "Conflict of field url on record eyjafjallajokull"
    "info": "https://server/docs/api.html#errors",
    "details": {
        "field": "url",
        "record": {
            "id": "eyjafjallajokull",
            "last_modified": 1430140411480,
            "url": "http://mozilla.org"
        }
    }
}
```

3.8.5 Validation errors

When multiple validation errors occur on a request, the first one is presented in the message.

The full list of validation errors is provided in the `details` field.

```
{
    "code": 400,
    "errno": 109,
    "error": "Bad Request",
    "message": "Invalid posted data",
    "info": "https://server/docs/api.html#errors",
    "details": [
        {
            "description": "42 is not a string: {'name': ''}",
            "location": "body",
            "name": "name"
        }
    ]
}
```

3.9 Deprecation

A track of the client version will be kept to know after which date each old version can be shutdown.

The date of the end of support is provided in the API root URL (e.g. `/v0`)

Using the `Alert` response header, the server can communicate any potential warning messages, information, or other alerts.

The value is JSON mapping with the following attributes:

- `code`: one of the strings `"soft-eol"` or `"hard-eol"`;
- `message`: a human-readable message (optional);
- `url`: a URL at which more information is available (optional).

A `410 Gone` error response can be returned if the client version is too old, or the service had been remplaced with a new and better service using a new protocol version.

See details in *Configuration* to activate deprecation.

CHAPTER 4

Internals

4.1 Installation

By default, a *Cliquet* application persists the records and cache in a local Redis.

Using the *application configuration*, other backends like « in-memory » or PostgreSQL can be enabled afterwards.

4.1.1 Supported Python versions

Cliquet supports Python 2.7, Python 3.4 and PyPy.

4.1.2 Distribute & Pip

Installing Cliquet with pip:

```
pip install cliquet
```

For *PostgreSQL* and *monitoring* support:

```
pip install cliquet[postgresql,monitoring]
```

Note: When installing cliquet with postgresql support in a virtualenv using the PyPy interpreter, the psycopg2cffi PostgreSQL database adapter will be installed, instead of the traditional psycopg2, as it provides significant performance improvements.

If everything is under control *python*-wise, jump to the next chapter. Otherwise please find more details below.

4.1.3 Python 3.4

Linux

```
sudo apt-get install python3.4-dev
```

OS X

```
brew install python3
```

4.1.4 Cryptography libraries

Linux

On Debian / Ubuntu based systems:

```
apt-get install libffi-dev libssl-dev
```

On RHEL-derivatives:

```
yum install libffi-devel openssl-devel
```

OS X

Assuming brew is installed:

```
brew install libffi openssl pkg-config
```

4.1.5 Install Redis

Linux

On debian / ubuntu based systems:

```
apt-get install redis-server
```

or:

```
yum install redis
```

OS X

Assuming brew is installed, Redis installation becomes:

```
brew install redis
```

To restart it (Bug after configuration update):

```
brew services restart redis
```

4.1.6 Install PostgreSQL

Client libraries only

Install PostgreSQL client headers:

```
sudo apt-get install libpq-dev
```

Install Cliquet with related dependencies:

```
pip install cliquet[postgresql]
```

Full server

PostgreSQL version 9.4 (or higher) is required.

To install PostgreSQL on Ubuntu/Debian use:

```
sudo apt-get install postgresql-9.4
```

If your Ubuntu/Debian distribution doesn't include version 9.4 of PostgreSQL look at the PostgreSQL Ubuntu and PostgreSQL Debian pages. The PostgreSQL project provides an Apt Repository that one can use to install recent PostgreSQL versions.

By default, the `postgres` user has no password and can hence only connect if ran by the `postgres` system user. The following command will assign it:

```
sudo -u postgres psql -c "ALTER USER postgres PASSWORD 'postgres';"
```

Cliquet requires `UTC` to be used as the database timezone, and `UTF-8` as the database encoding. You can for example use the following commands to create a database named `testdb` with the appropriate timezone and encoding:

```
sudo -u postgres psql -c "ALTER ROLE postgres SET TIMEZONE TO 'UTC';"
sudo -u postgres psql -c "CREATE DATABASE testdb ENCODING 'UTF-8';"
```

Server using Docker

Install docker, for example on Ubuntu:

```
sudo apt-get install docker.io
```

Run the official PostgreSQL container locally:

```
postgres=$(sudo docker run -d -p 5432:5432 postgres)
```

(*optional*) Create the test database:

```
psql -h localhost -U postgres -W
#> CREATE DATABASE "testdb";
```

Tag and save the current state with:

```
sudo docker commit $postgres cliquet-empty
```

In the future, run the tagged version of the container

```
cliquet=$(sudo docker run -d -p 5432:5432 cliquet-empty)

...

sudo docker stop $cliquet
```

4.2 Configuration

See Pyramid settings documentation.

4.2.1 Environment variables

In order to ease deployment or testing strategies, *Cliquet* reads settings from environment variables, in addition to `.ini` files.

For example, `cliquet.storage_backend` is read from environment variable `CLIQUET_STORAGE_BACKEND` if defined, else from application `.ini`, else from internal defaults.

4.2.2 Project info

```
cliquet.project_name = project
cliquet.project_docs = https://project.rtfd.org/
# cliquet.project_version = 1.3-stable
# cliquet.http_api_version = 1.0
```

It can be useful to set the `project_version` to a custom string, in order to prevent disclosing information about the currently running version (when there are known vulnerabilities for example).

4.2.3 Feature settings

```
# Limit number of batch operations per request
# cliquet.batch_max_requests = 25

# Force pagination *(recommended)*
# cliquet.paginate_by = 200

# Custom record id generator class
# cliquet.id_generator = cliquet.storage.generators.UUID4
```

4.2.4 Disabling endpoints

It is possible to deactivate specific resources operations, directly in the settings.

To do so, a setting key must be defined for the disabled resources endpoints:

```
'cliquet.{endpoint_type}_{resource_name}_{method}_enabled'
```

Where: - **endpoint_type** is either collection or record; - **resource_name** is the name of the resource (by default, *Cliquet* uses the name of the class); - **method** is the http method (in lower case): For instance `put`.

For instance, to disable the PUT on records for the *Mushrooms* resource, the following setting should be declared in the `.ini` file:

```
# Disable article collection DELETE endpoint
cliquet.collection_article_delete_enabled = false

# Disable mushroom record PATCH endpoint
cliquet.record_mushroom_patch_enabled = false
```

Setting the service in readonly mode

It is also possible to deploy a *Cliquet* service in readonly mode.

Instead of having settings to disable every resource endpoint, the `readonly` setting can be set:

```
cliquet.readonly = true
```

This will disable every resources endpoints that are not accessed with one of `GET`, `OPTIONS`, or `HEAD` methods. Requests will receive a `405 Method not allowed` error response.

This setting will also activate readonly heartbeat checks for the permission and the storage backend.

Warning: The cache backend will still needs read-write privileges, in order to cache OAuth authentication states and tokens for example.
If you do not need cache at all, set the `kinto.cache_backend` setting to an empty string to disable it.

4.2.5 Deployment

```
# cliquet.backoff = 10
cliquet.retry_after_seconds = 30
```

Scheme, host and port

By default *Cliquet* does not enforce requests scheme, host and port. It relies on WSGI specification and the related stack configuration. Tuning this becomes necessary when the application runs behind proxies or load balancers.

Most implementations, like *uwsgi*, provide configuration variables to adjust it properly.

However if, for some reasons, this had to be enforced at the application level, the following settings can be set:

```
# cliquet.http_scheme = https
# cliquet.http_host = production.server:7777
```

Check the `url` value returned in the hello view.

Deprecation

Activate the *service deprecation*. If the date specified in `eos` is in the future, an alert will be sent to clients. If it's in the past, the service will be declared as decomissionned.

```
# cliquet.eos = 2015-01-22
# cliquet.eos_message = "Client is too old"
# cliquet.eos_url = http://website/info-shutdown.html
```

Logging with Heka

Mozilla Services standard logging format can be enabled using:

```
cliquet.logging_renderer = cliquet.logs.MozillaHekaRenderer
```

With the following configuration, all logs are redirected to standard output (See 12factor app):

```
[loggers]
keys = root

[handlers]
keys = console

[formatters]
keys = heka

[logger_root]
level = INFO
handlers = console
formatter = heka

[handler_console]
class = StreamHandler
args = (sys.stdout,)
level = NOTSET

[formatter_heka]
format = %(message)s
```

Handling exceptions with Sentry

Requires the `raven` package, or *Cliquet* installed with `pip install cliquet[monitoring]`.

Sentry logging can be enabled, as explained in official documentation.

Note: The application sends an *INFO* message on startup, mainly for setup check.

Monitoring with StatsD

Requires the `statsd` package, or *Cliquet* installed with `pip install cliquet[monitoring]`.

StatsD metrics can be enabled (disabled by default):

```
cliquet.statsd_url = udp://localhost:8125
# cliquet.statsd_prefix = cliquet.project_name
```

Monitoring with New Relic

Requires the `newrelic` package, or *Cliquet* installed with `pip install cliquet[monitoring]`.

Enable middlewares as described *here*.

New-Relic can be enabled (disabled by default):

```
cliquet.newrelic_config = /location/of/newrelic.ini
cliquet.newrelic_env = prod
```

4.2.6 Storage

```
cliquet.storage_backend = cliquet.storage.redis
cliquet.storage_url = redis://localhost:6379/1

# Safety limit while fetching from storage
# cliquet.storage_max_fetch_size = 10000

# Control number of pooled connections
# cliquet.storage_pool_size = 50
```

See *storage backend documentation* for more details.

4.2.7 Notifications

To activate event listeners, use the *event_handlers* setting, which takes a list of either:

- aliases (e.g. `journal`)
- python modules (e.g. `cliquet.listeners.redis`)

Each listener will load load its dedicated settings.

In the example below, the Redis listener is activated and will send data in the `queue` Redis list.

```
cliquet.event_listeners = redis

cliquet.event_listeners.redis.use = cliquet.listeners.redis
cliquet.event_listeners.redis.url = redis://localhost:6379/0
cliquet.event_listeners.redis.pool_size = 5
cliquet.event_listeners.redis.listname = queue
```

Filtering

It is possible to filter events by action and/or resource name. By default actions `create`, `update` and `delete` are notified for every resources.

```
cliquet.event_listeners.redis.actions = create
cliquet.event_listeners.redis.resources = article comment
```

4.2.8 Cache

Backend

```
cliquet.cache_backend = cliquet.cache.redis
cliquet.cache_url = redis://localhost:6379/0
cliquet.cache_prefix = stack1_

# Control number of pooled connections
# cliquet.storage_pool_size = 50
```

See *cache backend documentation* for more details.

Headers

It is possible to add cache control headers on a particular resource for anonymous requests. The client (or proxy) will use them to cache the resource responses for a certain amount of time.

By default, *Cliquet* indicates the clients to invalidate their cache (`Cache-Control:  no-cache`).

```
cliquet.mushroom_cache_expires_seconds = 3600
```

Basically, this will add both `Cache-Control:  max-age=3600` and `Expire:  <server datetime + 1H>` response headers to the `GET` responses.

If setting is set to `0`, then the resource follows the default behaviour.

CORS

By default, CORS headers are cached by clients during 1H (`Access-Control-Max-Age`).

The duration can be set from settings. If set to empty or to 0, the header is not sent to clients.

```
cliquet.cors_max_age_seconds = 7200
```

4.2.9 Authentication

Since user identification is hashed in storage, a secret key is required in configuration:

```
# cliquet.userid_hmac_secret = b4c96a8692291d88fe5a97dd91846eb4
```

Authentication setup

Cliquet relies on pyramid multiauth to initialize authentication.

Therefore, any authentication policy can be specified through configuration.

For example, using the following example, *Basic Auth*, *Persona* and *IP Auth* are enabled:

```
multiauth.policies = basicauth pyramid_persona ipauth

multiauth.policy.ipauth.use = pyramid_ipauth.IPAuthentictionPolicy
multiauth.policy.ipauth.ipaddrs = 192.168.0.*
multiauth.policy.ipauth.userid = LAN-user
multiauth.policy.ipauth.principals = trusted
```

Similarly, any authorization policies and group finder function can be specified through configuration in order to deeply customize permissions handling and authorizations.

Basic Auth

`basicauth` is mentioned among `multiauth.policies` by default.

```
multiauth.policies = basicauth
```

By default, it uses an internal *Basic Auth* policy bundled with *Cliquet*.

In order to replace it by another one:

```
multiauth.policies = basicauth
multiauth.policy.basicauth.use = myproject.authn.BasicAuthPolicy
```

Custom Authentication

Using the various Pyramid authentication packages, it is possible to plug any kind of authentication.

(*Github/Twitter example to be done*)

Firefox Accounts

Enabling *Firefox Accounts* consists in including `cliquet_fxa` in configuration, mentioning `fxa` among policies and providing appropriate values for OAuth2 client settings.

See mozilla-services/cliquet-fxa.

4.2.10 Permissions

Backend

```
cliquet.permission_backend = cliquet.permission.redis
cliquet.permission_url = redis://localhost:6379/1

# Control number of pooled connections
# cliquet.permission_pool_size = 50
```

See *permission backend documentation* for more details.

Resources

ACEs are usually set on objects using the permission backend.

It is also possible to configure them from settings, and it will **bypass** the permission backend.

For example, for a resource named "bucket", the following setting will enable authenticated people to create bucket records:

```
cliquet.bucket_create_principals = system.Authenticated
```

The format of these permission settings is `<resource_name>_<permission>_principals = comma,separated,principals`.

See *shareable resource documentation* for more details.

4.2.11 Application profiling

It is possible to profile the application while its running. This is especially useful when trying to find slowness in the application.

Enable middlewares as described *here*.

Update the configuration file with the following values:

```
cliquet.profiler_enabled = true
cliquet.profiler_dir = /tmp/profiling
```

Run a load test (*for example*):

```
SERVER_URL=http://localhost:8000 make bench -e
```

Render execution graphs using GraphViz:

```
sudo apt-get install graphviz
```

```
pip install gprof2dot
gprof2dot -f pstats POST.v1.batch.000176ms.1427458675.prof | dot -Tpng -o output.png
```

4.2.12 Enable middleware

In order to enable Cliquet middleware, wrap the application in the project `main` function:

```
def main(global_config, **settings):
    config = Configurator(settings=settings)
    cliquet.initialize(config, __version__)
    app = config.make_wsgi_app()
    return cliquet.install_middlewares(app, settings)
```

4.2.13 Initialization sequence

In order to control what part of *Cliquet* should be run during application startup, or add custom initialization steps from configuration, it is possible to change the `initialization_sequence` setting.

Warning: This is considered as a dangerous zone and should be used with caution.
Later, a better formalism should be introduced to easily allow addition or removal of steps, without repeating the whole list and without relying on internal functions location.

```
cliquet.initialization_sequence = cliquet.initialization.setup_request_bound_data
                                  cliquet.initialization.setup_json_serializer
                                  cliquet.initialization.setup_logging
                                  cliquet.initialization.setup_storage
                                  cliquet.initialization.setup_permission
                                  cliquet.initialization.setup_cache
                                  cliquet.initialization.setup_requests_scheme
                                  cliquet.initialization.setup_version_redirection
                                  cliquet.initialization.setup_deprecation
                                  cliquet.initialization.setup_authentication
                                  cliquet.initialization.setup_backoff
                                  cliquet.initialization.setup_statsd
                                  cliquet.initialization.setup_listeners
                                  cliquet.events.setup_transaction_hook
```

4.3 Resource

Cliquet provides a basic component to build resource oriented APIs. In most cases, the main customization consists in defining the schema of the records for this resource.

4.3.1 Full example

```
import colander

from cliquet import resource
from cliquet import utils


class BookmarkSchema(resource.ResourceSchema):
    url = colander.SchemaNode(colander.String(), validator=colander.url)
    title = colander.SchemaNode(colander.String())
    favorite = colander.SchemaNode(colander.Boolean(), missing=False)
    device = colander.SchemaNode(colander.String(), missing='')

    class Options:
        readonly_fields = ('device',)
        unique_fields = ('url',)


@resource.register()
class Bookmark(resource.UserResource):
    mapping = BookmarkSchema()

    def process_record(self, new, old=None):
        new = super(Bookmark, self).process_record(new, old)
        if new['device'] != old['device']:
            new['device'] = self.request.headers.get('User-Agent')

        return new
```

See the ReadingList and Kinto projects source code for real use cases.

4.3.2 URLs

By default, a resource defines two URLs:

- `/{classname}s` for the list of records
- `/{classname}s/{id}` for single records

Since adding an `s` suffix for the plural form might not always be relevant, URLs can be specified during registration:

```
@resource.register(collection_path='/user/bookmarks',
                   record_path='/user/bookmarks/{{id}}')
class Bookmark(resource.UserResource):
    mapping = BookmarkSchema()
```

Note: The same resource can be registered with different URLs.

4.3.3 Schema

Override the base schema to add extra fields using the Colander API.

```
class Movie(ResourceSchema):
    director = colander.SchemaNode(colander.String())
    year = colander.SchemaNode(colander.Int(),
                               validator=colander.Range(min=1850))
    genre = colander.SchemaNode(colander.String(),
                                validator=colander.OneOf(['Sci-Fi', 'Comedy']))
```

See the *resource schema options* to define *schema-less* resources or specify rules for unicity or readonly.

4.3.4 Permissions

Using the `cliquet.resource.UserResource`, the resource is accessible by any authenticated request, but the records are isolated by *user id*.

In order to define resources whose records are not isolated, open publicly or controlled with individual fined-permissions, a `cliquet.resource.ShareableResource` could be used.

But there are other strategies, please refer to *dedicated section about permissions*.

4.3.5 HTTP methods and options

In order to specify which HTTP verbs (`GET`, `PUT`, `PATCH`, ...) are allowed on the resource, as well as specific custom Pyramid (or cornice) view arguments, refer to the *viewset section*.

4.3.6 Events

When a record is created/deleted in a resource, an event is sent. See the dedicated section about notifications to plug events in your Pyramid/*Cliquet* application or plugin.

4.3.7 Model

Plug custom model

In order to customize the interaction of a HTTP resource with its storage, a custom model can be plugged-in:

```
from cliquet import resource


class TrackedModel(resource.Model):
    def create_record(self, record, parent_id=None, unique_fields=None):
        record = super(TrackedModel, self).create_record(record,
                                                         parent_id,
                                                         unique_fields)
        trackid = index.track(record)
        record['trackid'] = trackid
        return record


class Payment(resource.UserResource):
    default_model = TrackedModel
```

Relationships

With the default model and storage backend, *Cliquet* does not support complex relations.

However, it is possible to plug a custom *model class*, that will take care of saving and retrieving records with relations.

Note: This part deserves more love, please come and discuss!

In Pyramid views

In Pyramid views, a `request` object is available and allows to use the storage configured in the application:

```
from cliquet import resource

def view(request):
    registry = request.registry

    flowers = resource.Model(storage=registry.storage,
                             collection_id='app:flowers')

    flowers.create_record({'name': 'Jonquille', 'size': 30})
    flowers.create_record({'name': 'Amapola', 'size': 18})

    min_size = resource.Filter('size', 20, resource.COMPARISON.MIN)
    records, total = flowers.get_records(filters=[min_size])

    flowers.delete_record(records[0])
```

Outside views

Outside views, an application context has to be built from scratch.

As an example, let's build a code that will copy a collection into another:

```
from cliquet import resource, DEFAULT_SETTINGS
from pyramid import Configurator


config = Configurator(settings=DEFAULT_SETTINGS)
config.add_settings({
    'cliquet.storage_backend': 'cliquet.storage.postgresql'
    'cliquet.storage_url': 'postgres://user:pass@db.server.lan:5432/dbname'
})
cliquet.initialize(config, '0.0.1')

local = resource.Model(storage=config.registry.storage,
                       parent_id='browsing',
                       collection_id='history')

remote = resource.Model(storage=config_remote.registry.storage,
                        parent_id='',
                        collection_id='history')

records, total = in remote.get_records():
```

```
for record in records:
    local.create_record(record)
```

4.3.8 Custom record ids

By default, records ids are UUID4.

A custom record ID generator can be set globally in *Configuration*, or at the resource level:

```
from cliquet import resource
from cliquet import utils
from cliquet.storage import generators


class MsecId(generators.Generator):
    def __call__(self):
        return '%s' % utils.msec_time()


@resource.register()
class Mushroom(resource.UserResource):
    def __init__(request):
        super(Mushroom, self).__init__(request)
        self.model.id_generator = MsecId()
```

4.3.9 Python API

Resource

class `cliquet.resource.`**`UserResource`**(*request*, *context=None*)

Base resource class providing every endpoint.

`default_viewset`

Default `cliquet.viewset.ViewSet` class to use when the resource is registered.

alias of `ViewSet`

`default_model`

Default `cliquet.resource.model.Model` class to use for interacting the *`cliquet.storage`* and `cliquet.permission` backends.

alias of `Model`

`mapping` = <cliquet.resource.schema.ResourceSchema object at 140435170666000 (named)>

Schema to validate records.

`collection`

The collection property.

`get_parent_id`(*request*)

Return the parent_id of the resource with regards to the current request.

Parameters **`request`** – The request used to create the resource.

Return type str

`is_known_field`(*field*)

Return `True` if *field* is defined in the resource mapping.

Parameters **field** (*str*) – Field name

Return type bool

collection_get()

Model `GET` endpoint: retrieve multiple records.

Raises `HTTPNotModified` if `If-None-Match` header is provided and collection not modified in the interim.

Raises `HTTPPreconditionFailed` if `If-Match` header is provided and collection modified in the iterim.

Raises `HTTPBadRequest` if filters or sorting are invalid.

collection_post()

Model `POST` endpoint: create a record.

If the new record conflicts against a unique field constraint, the posted record is ignored, and the existing record is returned, with a `200` status.

Raises `HTTPPreconditionFailed` if `If-Match` header is provided and collection modified in the iterim.

See also:

Add custom behaviour by overriding *cliquet.resource.UserResource.process_record()*

collection_delete()

Model `DELETE` endpoint: delete multiple records.

Raises `HTTPPreconditionFailed` if `If-Match` header is provided and collection modified in the iterim.

Raises `HTTPBadRequest` if filters are invalid.

get()

Record `GET` endpoint: retrieve a record.

Raises `HTTPNotFound` if the record is not found.

Raises `HTTPNotModified` if `If-None-Match` header is provided and record not modified in the interim.

Raises `HTTPPreconditionFailed` if `If-Match` header is provided and record modified in the iterim.

put()

Record `PUT` endpoint: create or replace the provided record and return it.

Raises `HTTPPreconditionFailed` if `If-Match` header is provided and record modified in the iterim.

Note: If `If-None-Match: *` request header is provided, the `PUT` will succeed only if no record exists with this id.

See also:

Add custom behaviour by overriding *cliquet.resource.UserResource.process_record()*.

patch()

Record `PATCH` endpoint: modify a record and return its new version.

If a request header `Response-Behavior` is set to `light`, only the fields whose value was changed are returned. If set to `diff`, only the fields whose value became different than the one provided are returned.

Raises `HTTPNotFound` if the record is not found.

Raises `HTTPPreconditionFailed` if `If-Match` header is provided and record modified in the iterim.

See also:

Add custom behaviour by overriding *`cliquet.resource.UserResource.apply_changes()`* or *`cliquet.resource.UserResource.process_record()`*.

`delete`()

Record `DELETE` endpoint: delete a record and return it.

Raises `HTTPNotFound` if the record is not found.

Raises `HTTPPreconditionFailed` if `If-Match` header is provided and record modified in the iterim.

`process_record`(*new*, *old=None*)

Hook for processing records before they reach storage, to introduce specific logics on fields for example.

```
def process_record(self, new, old=None):
    new = super(MyResource, self).process_record(new, old)
    version = old['version'] if old else 0
    new['version'] = version + 1
    return new
```

Or add extra validation based on request:

```
from cliquet.errors import raise_invalid

def process_record(self, new, old=None):
    new = super(MyResource, self).process_record(new, old)
    if new['browser'] not in request.headers['User-Agent']:
        raise_invalid(self.request, name='browser', error='Wrong')
    return new
```

Parameters

- **`new`** (*`dict`*) – the validated record to be created or updated.
- **`old`** (*`dict`*) – the old record to be updated, `None` for creation endpoints.

Returns the processed record.

Return type dict

`apply_changes`(*record*, *changes*)

Merge *changes* into *record* fields.

Note: This is used in the context of PATCH only.

Override this to control field changes at record level, for example:

```
def apply_changes(self, record, changes):
    # Ignore value change if inferior
    if record['position'] > changes.get('position', -1):
        changes.pop('position', None)
    return super(MyResource, self).apply_changes(record, changes)
```

Raises `HTTPBadRequest` if result does not comply with resource schema.

Returns the new record with *changes* applied.

Return type dict

Schema

class `cliquet.resource.schema.`**`ResourceSchema`**`(`**arg*, ***kw*`)`

Base resource schema, with *Cliquet* specific built-in options.

class **`Options`**

Resource schema options.

This is meant to be overriden for changing values:

```
class Product(ResourceSchema):
    reference = colander.SchemaNode(colander.String())

    class Options:
        unique_fields = ('reference',)
```

`unique_fields` = ()

Fields that must have unique values for the user collection. During records creation and modification, a conflict error will be raised if unicity is about to be violated.

`readonly_fields` = ()

Fields that cannot be updated. Values for fields will have to be provided either during record creation, through default values using `missing` attribute or implementing a custom logic in *`cliquet.resource.UserResource.process_record()`*.

`preserve_unknown` = False

Define if unknown fields should be preserved or not.

For example, in order to define a schema-less resource, in other words a resource that will accept any form of record, the following schema definition is enough:

```
class SchemaLess(ResourceSchema):
    class Options:
        preserve_unknown = True
```

`ResourceSchema.`**`is_readonly`**`(`*field*`)`

Return True if specified field name is read-only.

Parameters **`field`** (*`str`*) – the field name in the schema

Returns `True` if the specified field is read-only, `False` otherwise.

Return type bool

class `cliquet.resource.schema.`**`PermissionsSchema`**`(`**args*, ***kwargs*`)`

A permission mapping defines ACEs.

It has permission names as keys and principals as values.

```
    {
        "write": ["fxa:af3e077eb9f5444a949ad65aa86e82ff"],
        "groups:create": ["fxa:70a9335eecfe440fa445ba752a750f3d"]
    }
```

class `cliquet.resource.schema.`**`TimeStamp`**(**arg*, ***kw*)

Basic integer schema field that can be set to current server timestamp in milliseconds if no value is provided.

```
    class Book(ResourceSchema):
        added_on = TimeStamp()
        read_on = TimeStamp(auto_now=False, missing=-1)
```

`schema_type`

alias of `Integer`

`title` **= 'Epoch timestamp'**

Default field title.

`auto_now` **= True**

Set to current server timestamp (*milliseconds*) if not provided.

`missing` **= None**

Default field value if not provided in record.

class `cliquet.resource.schema.`**`URL`**(**arg*, ***kw*)

String field representing a URL, with max length of 2048. This is basically a shortcut for string field with *~colander:colander.url*.

```
    class BookmarkSchema(ResourceSchema):
        url = URL()
```

`schema_type`

alias of `String`

Model

class `cliquet.resource.`**`Model`**(*storage*, *id_generator=None*, *collection_id=''*, *parent_id=''*, *auth=None*)

A collection stores and manipulate records in its attached storage.

It is not aware of HTTP environment nor protocol.

Records are isolated according to the provided *name* and *parent_id*.

Those notions have no particular semantic and can represent anything. For example, the collection *name* can be the *type* of objects stored, and *parent_id* can be the current *user id* or *a group* where the collection belongs. If left empty, the collection records are not isolated.

`id_field` **= 'id'**

Name of *id* field in records

`modified_field` **= 'last_modified'**

Name of *last modified* field in records

`deleted_field` **= 'deleted'**

Name of *deleted* field in deleted records

`timestamp`(*parent_id=None*)

Fetch the collection current timestamp.

Parameters **`parent_id`** (`str`) – optional filter for parent id

Return type integer

get_records (*filters=None*, *sorting=None*, *pagination_rules=None*, *limit=None*, *include_deleted=False*, *parent_id=None*)

Fetch the collection records.

Override to post-process records after feching them from storage.

Parameters

- **filters** (list of *cliquet.storage.Filter*) – Optionally filter the records by their attribute. Each filter in this list is a tuple of a field, a value and a comparison (see *cliquet.utils.COMPARISON*). All filters are combined using *AND*.
- **sorting** (list of *cliquet.storage.Sort*) – Optionnally sort the records by attribute. Each sort instruction in this list refers to a field and a direction (negative means descending). All sort instructions are cumulative.
- **pagination_rules** (list of list of *cliquet.storage.Filter*) – Optionnally paginate the list of records. This list of rules aims to reduce the set of records to the current page. A rule is a list of filters (see *filters* parameter), and all rules are combined using *OR*.
- **limit** (*int*) – Optionnally limit the number of records to be retrieved.
- **include_deleted** (*bool*) – Optionnally include the deleted records that match the filters.
- **parent_id** (*str*) – optional filter for parent id

Returns A tuple with the list of records in the current page, the total number of records in the result set.

Return type tuple

delete_records (*filters=None*, *parent_id=None*)

Delete multiple collection records.

Override to post-process records after their deletion from storage.

Parameters

- **filters** (list of *cliquet.storage.Filter*) – Optionally filter the records by their attribute. Each filter in this list is a tuple of a field, a value and a comparison (see *cliquet.utils.COMPARISON*). All filters are combined using *AND*.
- **parent_id** (*str*) – optional filter for parent id

Returns The list of deleted records from storage.

get_record (*record_id*, *parent_id=None*)

Fetch current view related record, and raise 404 if missing.

Parameters

- **record_id** (*str*) – record identifier
- **parent_id** (*str*) – optional filter for parent id

Returns the record from storage

Return type dict

create_record (*record*, *parent_id=None*, *unique_fields=None*)

Create a record in the collection.

Override to perform actions or post-process records after their creation in storage.

```
    def create_record(self, record):
        record = super(MyModel, self).create_record(record)
        idx = index.store(record)
        record['index'] = idx
        return record
```

Parameters

- **record** (*dict*) – record to store
- **parent_id** (*str*) – optional filter for parent id
- **unique_fields** (*tuple*) – list of fields that should remain unique

Returns the newly created record.

Return type dict

update_record(*record*, *parent_id=None*, *unique_fields=None*)

Update a record in the collection.

Override to perform actions or post-process records after their modification in storage.

```
    def update_record(self, record, parent_id=None,unique_fields=None):
        record = super(MyModel, self).update_record(record,
                                                    parent_id,
                                                    unique_fields)
        subject = 'Record {} was changed'.format(record[self.id_field])
        send_email(subject)
        return record
```

Parameters

- **record** (*dict*) – record to store
- **parent_id** (*str*) – optional filter for parent id
- **unique_fields** (*tuple*) – list of fields that should remain unique

Returns the updated record.

Return type dict

delete_record(*record*, *parent_id=None*, *last_modified=None*)

Delete a record in the collection.

Override to perform actions or post-process records after deletion from storage for example:

```
    def delete_record(self, record):
        deleted = super(MyModel, self).delete_record(record)
        erase_media(record)
        deleted['media'] = 0
        return deleted
```

Parameters

- **record** (*dict*) – the record to delete
- **record** – record to store
- **parent_id** (*str*) – optional filter for parent id

Returns the deleted record.

Return type dict

Generators

class `cliquet.storage.generators.`**`Generator`**`(`*config=None*`)`

Base generator for records ids.

Id generators are used by storage backend during record creation, and at resource level to validate record id in requests paths.

`regexp` **= '^[a-zA-Z0-9][a-zA-Z0-9_-]*$'**

Default record id pattern. Can be changed to comply with custom ids.

`match``(`*record_id*`)`

Validate that record ids match the generator. This is used mainly when a record id is picked arbitrarily (e.g with `PUT` requests).

Returns *True* if the specified record id matches expected format.

Return type bool

class `cliquet.storage.generators.`**`UUID4`**`(`*config=None*`)`

UUID4 record id generator.

UUID block are separated with `-`. (example: `'472be9ec-26fe-461b-8282-9c4e4b207ab3'`)

UUIDs are very safe in term of unicity. If 1 billion of UUIDs are generated every second for the next 100 years, the probability of creating just one duplicate would be about 50% (source).

`regexp` **= '^[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}$'**

UUID4 accurate pattern.

4.4 Viewsets

Cliquet maps URLs, *endpoints* and *permissions* to resources using *ViewSets*.

Since a resource defines two URLs with several HTTP methods, a view set can be considered as a set of rules for registring the resource views into the routing mechanism of Pyramid.

To use *Cliquet* in a basic fashion, there is no need to understand how viewsets work in full detail.

4.4.1 Override defaults

Viewsets defaults can be overriden by passing arguments to the `cliquet.resource.register()` class decorator:

```python
from cliquet import resource


@resource.register(collection_methods=('GET',))
class Resource(resource.UserResource):
    mapping = BookmarkSchema()
```

4.4.2 Subclassing

In case this isn't enough, the `cliquet.resource.viewset.ViewSet` class can be subclassed and specified during registration:

```
from cliquet import resource


class NoSchemaViewSet(resource.ViewSet):

    def get_record_schema(self, resource_cls, method):
        simple_mapping = colander.MappingSchema(unknown='preserve')
        return simple_mapping


@resource.register(viewset=NoSchemaViewSet())
class Resource(resource.UserResource):
    mapping = BookmarkSchema()
```

4.4.3 ViewSet class

class `cliquet.resource.`**`ViewSet`**(***kwargs*)

The default ViewSet object.

A viewset contains all the information needed to register any resource in the Cornice registry.

It provides the same features as `cornice.resource()`, except that it is much more flexible and extensible.

`update`(***kwargs*)

Update viewset attributes with provided values.

`get_view_arguments`(*endpoint_type*, *resource_cls*, *method*)

Return the Pyramid/Cornice view arguments for the given endpoint type and method.

Parameters

- **`endpoint_type`** (*`str`*) – either "collection" or "record".
- **`resource_cls`** – the resource class.
- **`method`** (*`str`*) – the HTTP method.

`get_record_schema`(*resource_cls*, *method*)

Return the Cornice schema for the given method.

`get_view`(*endpoint_type*, *method*)

Return the view method name located on the resource object, for the given type and method.

•For collections, this will be "collection_{method|lower}

•For records, this will be "{method|lower}.

`get_name`(*resource_cls*)

Returns the name of the resource.

`get_service_name`(*endpoint_type*, *resource_cls*)

Returns the name of the service, depending a given type and resource.

`is_endpoint_enabled`(*endpoint_type*, *resource_name*, *method*, *settings*)

Returns if the given endpoint is enabled or not.

Uses the settings to tell so.

4.5 Storage

4.5.1 Backends

PostgreSQL

class `cliquet.storage.postgresql.`**`Storage`**(*client*, *max_fetch_size*, **args*, ***kwargs*)

Storage backend using PostgreSQL.

Recommended in production (*requires PostgreSQL 9.4 or higher*).

Enable in configuration:

```
cliquet.storage_backend = cliquet.storage.postgresql
```

Database location URI can be customized:

```
cliquet.storage_url = postgres://user:pass@db.server.lan:5432/dbname
```

Alternatively, username and password could also rely on system user ident or even specified in `~/.pgpass` (*see PostgreSQL documentation*).

Note: Some tables and indices are created when `cliquet migrate` is run. This requires some privileges on the database, or some error will be raised.

Alternatively, the schema can be initialized outside the python application, using the SQL file located in `cliquet/storage/postgresql/schema.sql`. This allows to distinguish schema manipulation privileges from schema usage.

A connection pool is enabled by default:

```
cliquet.storage_pool_size = 10
cliquet.storage_maxoverflow = 10
cliquet.storage_max_backlog = -1
cliquet.storage_pool_recycle = -1
cliquet.storage_pool_timeout = 30
cliquet.cache_poolclass = cliquet.storage.postgresql.pool.QueuePoolWithMaxBacklog
```

The `max_backlog` limits the number of threads that can be in the queue waiting for a connection. Once this limit has been reached, any further attempts to acquire a connection will be rejected immediately, instead of locking up all threads by keeping them waiting in the queue.

See dedicated section in SQLAlchemy documentation for default values and behaviour.

Note: Using a dedicated connection pool is still recommended to allow load balancing, replication or limit the number of connections used in a multi-process deployment.

Redis

class `cliquet.storage.redis.`**`Storage`**(*client*, **args*, ***kwargs*)

Storage backend implementation using Redis.

Warning: Useful for very low server load, but won't scale since records sorting and filtering are performed in memory.

Enable in configuration:

```
cliquet.storage_backend = cliquet.storage.redis
```

(Optional) Instance location URI can be customized:

```
cliquet.storage_url = redis://localhost:6379/0
```

A threaded connection pool is enabled by default:

```
cliquet.storage_pool_size = 50
```

Memory

class `cliquet.storage.memory.`**`Storage`**(**args*, ***kwargs*)

Storage backend implementation in memory.

Useful for development or testing purposes, but records are lost after each server restart.

Enable in configuration:

```
cliquet.storage_backend = cliquet.storage.memory
```

4.5.2 API

Implementing a custom storage backend consists in implementating the following interface:

class `cliquet.storage.`**`Filter`**(*field*, *value*, *operator*)

Filtering properties.

`field`
Alias for field number 0

`operator`
Alias for field number 2

`value`
Alias for field number 1

class `cliquet.storage.`**`Sort`**(*field*, *direction*)

Sorting properties.

`direction`
Alias for field number 1

`field`
Alias for field number 0

class `cliquet.storage.`**`StorageBase`**

Storage abstraction used by resource views.

It is meant to be instantiated at application startup. Any operation may raise a *HTTPServiceUnavailable* error if an error occurs with the underlying service.

Configuration can be changed to choose which storage backend will persist the objects.

Raises `HTTPServiceUnavailable`

initialize_schema()
Create every necessary objects (like tables or indices) in the backend.

This is excuted when the `cliquet migrate` command is ran.

flush(*auth=None*)
Remove **every** object from this storage.

collection_timestamp(*collection_id*, *parent_id*, *auth=None*)
Get the highest timestamp of every objects in this *collection_id* for this *parent_id*.

Note: This should take deleted objects into account.

Parameters

- **collection_id** (*str*) – the collection id.
- **parent_id** (*str*) – the collection parent.

Returns the latest timestamp of the collection.

Return type int

create(*collection_id*, *parent_id*, *object*, *id_generator=None*, *unique_fields=None*, *id_field='id'*, *modified_field='last_modified'*, *auth=None*)
Create the specified *object* in this *collection_id* for this *parent_id*. Assign the id to the object, using the attribute `cliquet.resource.Model.id_field`.

Note: This will update the collection timestamp.

Raises `cliquet.storage.exceptions.UnicityError`

Parameters

- **collection_id** (*str*) – the collection id.
- **parent_id** (*str*) – the collection parent.
- **object** (*dict*) – the object to create.

Returns the newly created object.

Return type dict

get(*collection_id*, *parent_id*, *object_id*, *id_field='id'*, *modified_field='last_modified'*, *auth=None*)
Retrieve the object with specified *object_id*, or raise error if not found.

Raises `cliquet.storage.exceptions.RecordNotFoundError`

Parameters

- **collection_id** (*str*) – the collection id.
- **parent_id** (*str*) – the collection parent.
- **object_id** (*str*) – unique identifier of the object

Returns the object object.

Return type dict

update(*collection_id*, *parent_id*, *object_id*, *object*, *unique_fields=None*, *id_field='id'*, *modified_field='last_modified'*, *auth=None*)

Overwrite the *object* with the specified *object_id*.

If the specified id is not found, the object is created with the specified id.

Note: This will update the collection timestamp.

Raises `cliquet.storage.exceptions.UnicityError`

Parameters

- **collection_id** (*str*) – the collection id.
- **parent_id** (*str*) – the collection parent.
- **object_id** (*str*) – unique identifier of the object
- **object** (*dict*) – the object to update or create.

Returns the updated object.

Return type dict

delete(*collection_id*, *parent_id*, *object_id*, *with_deleted=True*, *id_field='id'*, *modified_field='last_modified'*, *deleted_field='deleted'*, *auth=None*)

Delete the object with specified *object_id*, and raise error if not found.

Deleted objects must be removed from the database, but their ids and timestamps of deletion must be tracked for synchronization purposes. (See `cliquet.storage.StorageBase.get_all()`)

Note: This will update the collection timestamp.

Raises `cliquet.storage.exceptions.RecordNotFoundError`

Parameters

- **collection_id** (*str*) – the collection id.
- **parent_id** (*str*) – the collection parent.
- **object_id** (*str*) – unique identifier of the object
- **with_deleted** (*bool*) – track deleted record with a tombstone

Returns the deleted object, with minimal set of attributes.

Return type dict

delete_all(*collection_id*, *parent_id*, *filters=None*, *with_deleted=True*, *id_field='id'*, *modified_field='last_modified'*, *deleted_field='deleted'*, *auth=None*)

Delete all objects in this *collection_id* for this *parent_id*.

Parameters

- **collection_id** (*str*) – the collection id.
- **parent_id** (*str*) – the collection parent.
- **filters** (list of `cliquet.storage.Filter`) – Optionnally filter the objects to delete.

- **`with_deleted`** (*`bool`*) – track deleted records with a tombstone

Returns the list of deleted objects, with minimal set of attributes.

Return type list of dict

`purge_deleted`(*collection_id*, *parent_id*, *before=None*, *id_field='id'*, *modified_field='last_modified'*, *auth=None*)

Delete all deleted object tombstones in this *collection_id* for this *parent_id*.

Parameters

- **`collection_id`** (*`str`*) – the collection id.
- **`parent_id`** (*`str`*) – the collection parent.
- **`before`** (*`int`*) – Optionnal timestamp to limit deletion (exclusive)

Returns The number of deleted objects.

Return type int

`get_all`(*collection_id*, *parent_id*, *filters=None*, *sorting=None*, *pagination_rules=None*, *limit=None*, *include_deleted=False*, *id_field='id'*, *modified_field='last_modified'*, *deleted_field='deleted'*, *auth=None*)

Retrieve all objects in this *collection_id* for this *parent_id*.

Parameters

- **`collection_id`** (*`str`*) – the collection id.
- **`parent_id`** (*`str`*) – the collection parent.
- **`filters`** (list of *`cliquet.storage.Filter`*) – Optionally filter the objects by their attribute. Each filter in this list is a tuple of a field, a value and a comparison (see *cliquet.utils.COMPARISON*). All filters are combined using *AND*.
- **`sorting`** (list of *`cliquet.storage.Sort`*) – Optionnally sort the objects by attribute. Each sort instruction in this list refers to a field and a direction (negative means descending). All sort instructions are cumulative.
- **`pagination_rules`** (list of list of *`cliquet.storage.Filter`*) – Optionnally paginate the list of objects. This list of rules aims to reduce the set of objects to the current page. A rule is a list of filters (see *filters* parameter), and all rules are combined using *OR*.
- **`limit`** (*`int`*) – Optionnally limit the number of objects to be retrieved.
- **`include_deleted`** (*`bool`*) – Optionnally include the deleted objects that match the filters.

Returns the limited list of objects, and the total number of matching objects in the collection (deleted ones excluded).

Return type tuple (list, integer)

Exceptions

Exceptions raised by storage backend.

exception `cliquet.storage.exceptions.`**`BackendError`**(*original=None*, *message=None*, **args*, ***kwargs*)

A generic exception raised by storage on error.

Parameters **`original`** (*`Exception`*) – the wrapped exception raised by underlying library.

exception `cliquet.storage.exceptions.`**`RecordNotFoundError`**

An exception raised when a specific record could not be found.

exception `cliquet.storage.exceptions.`**`UnicityError`**(*field*, *record*, **args*, ***kwargs*)

An exception raised on unicity constraint violation.

Raised by storage backend when the creation or the modification of a record violates the unicity constraints defined by the resource.

4.5.3 Store custom data

Storage can be used to store arbitrary data.

```
data = {'subscribed': datetime.now()}
user_id = request.authenticated_userid

storage = request.registry.storage
storage.create(collection_id='__custom', parent_id='', record=data)
```

See the *Model* class to manipulate collections of records.

4.6 Cache

4.6.1 PostgreSQL

class `cliquet.cache.postgresql.`**`Cache`**(*client*, **args*, ***kwargs*)

Cache backend using PostgreSQL.

Enable in configuration:

```
    cliquet.cache_backend = cliquet.cache.postgresql
```

Database location URI can be customized:

```
    cliquet.cache_url = postgres://user:pass@db.server.lan:5432/dbname
```

Alternatively, username and password could also rely on system user ident or even specified in `~/.pgpass` (*see PostgreSQL documentation*).

Note: Some tables and indices are created when `cliquet migrate` is run. This requires some privileges on the database, or some error will be raised.

Alternatively, the schema can be initialized outside the python application, using the SQL file located in `cliquet/cache/postgresql/schema.sql`. This allows to distinguish schema manipulation privileges from schema usage.

A connection pool is enabled by default:

```
    cliquet.cache_pool_size = 10
    cliquet.cache_maxoverflow = 10
    cliquet.cache_max_backlog = -1
    cliquet.cache_pool_recycle = -1
    cliquet.cache_pool_timeout = 30
    cliquet.cache_poolclass = cliquet.storage.postgresql.pool.QueuePoolWithMaxBacklog
```

The `max_backlog` limits the number of threads that can be in the queue waiting for a connection. Once this limit has been reached, any further attempts to acquire a connection will be rejected immediately, instead of locking up all threads by keeping them waiting in the queue.

See dedicated section in SQLAlchemy documentation for default values and behaviour.

Note: Using a dedicated connection pool is still recommended to allow load balancing, replication or limit the number of connections used in a multi-process deployment.

Noindex

4.6.2 Redis

class `cliquet.cache.redis.`**`Cache`**(*client*, **args*, ***kwargs*)

Cache backend implementation using Redis.

Enable in configuration:

```
cliquet.cache_backend = cliquet.cache.redis
```

(Optional) Instance location URI can be customized:

```
cliquet.cache_url = redis://localhost:6379/1
```

A threaded connection pool is enabled by default:

```
cliquet.cache_pool_size = 50
```

If the database is used for multiple Kinto deployement cache, you may want to add a prefix to every key to avoid collision:

```
cliquet.cache_prefix = stack1_
```

Noindex

4.6.3 Memory

class `cliquet.cache.memory.`**`Cache`**(**args*, ***kwargs*)

Cache backend implementation in local thread memory.

Enable in configuration:

```
cliquet.cache_backend = cliquet.cache.memory
```

Noindex

4.6.4 API

Implementing a custom cache backend consists on implementing the following interface:

class `cliquet.cache.`**`CacheBase`**(**args*, ***kwargs*)

initialize_schema()
Create every necessary objects (like tables or indices) in the backend.

This is excuted when the `cliquet migrate` command is ran.

flush()
Delete every values.

ttl(*key*)
Obtain the expiration value of the specified *key*.

Parameters **key** (*str*) – key

Returns number of seconds or negative if no TTL.

Return type float

expire(*key*, *ttl*)
Set the expiration value *ttl* for the specified *key*.

Parameters

- **key** (*str*) – key
- **ttl** (*float*) – number of seconds

set(*key*, *value*, *ttl=None*)
Store a value with the specified *key*. If *ttl* is provided, set an expiration value.

Parameters

- **key** (*str*) – key
- **value** (*str*) – value to store
- **ttl** (*float*) – expire after number of seconds

get(*key*)
Obtain the value of the specified *key*.

Parameters **key** (*str*) – key

Returns the stored value or None if missing.

Return type str

delete(*key*)
Delete the value of the specified *key*.

Parameters **key** (*str*) – key

4.7 Notifications

Knowing some records have been modified in a resource is very useful to integrate a Cliquet-based application with other services.

For example, a search service that gets notified everytime something has changed, can continuously update its index.

Cliquet leverages Pyramid's built-in event system and produces the following events:

- `cliquet.events.ResourceRead`: a read operation occured on the resource.

- `cliquet.events.ResourceChanged`: a resource **is being changed**. This event occurs synchronously within the transaction and within the request/response cycle. Commit is not yet done and rollback is still possible.

  Subscribers of this event are likely to perform database operations, alter the server response, or cancel the transaction (by raising an HTTP exception for example). Do not subscribe to this event for operations that will not be rolled-back automatically.

- `cliquet.events.AfterResourceChanged`: a resource **was changed** and **committed**.

  Subscribers of this event can fail, errors are swallowed and logged. The final transaction result (or response) cannot be altered.

  Subscribers of this event are likely to perform irreversible actions that requires data to be committed in database (like sending messages, deleting files on disk, or run asynchronous tasks).

Event subscribers can then pick up those events and act upon them.

```
from cliquet.events import AfterResourceChanged


def on_resource_changed(event):
    for change in event.impacted_records:
        start_download(change['new']['url'])

config.add_subscriber(on_resource_changed, AfterResourceChanged)
```

4.7.1 Transactions

Only one event is sent per transaction, per resource and per action.

In other words, if every requests of a *batch requests* perform the same action on the same resource, only one event will be sent.

The `AfterResourceChanged` is sent only if the transaction was comitted successfully.

It is possible to cancel the current transaction by raising an HTTP Exception from a `ResourceChanged` event. For example:

```
from cliquet.events import ResourceChanged
from pyramid import httpexceptions

def check_quota(event):
     max_quota = event.request.registry.settings['max_quota']
     if check_quota(event, max_quota):
        raise httpexceptions.HTTPInsufficientStorage()

config.add_subscriber(check_quota, ResourceChanged)
```

4.7.2 Filtering

It is possible to filter events based on its action or the name of the resource where it occured.

For example:

```
from cliquet.events import ResourceChanged, ACTIONS

config.add_subscriber(on_mushroom_changed, ResourceChanged, for_resources=('mushroom',))
config.add_subscriber(on_record_deleted, ResourceChanged, for_actions=(ACTIONS.DELETE,))
```

4.7.3 Payload

The `cliquet.events.ResourceChanged` and `cliquet.events.AfterResourceChanged` events contain a `payload` attribute with the following information:

- **timestamp**: the time of the event
- **action**: what happened. 'create', 'update' or 'delete'
- **uri**: the uri of the impacted resource
- **user_id**: the authenticated user id
- **resource_name**: the name of the impacted resouce (e.g. 'article', 'bookmark', bucket', 'group' etc.)
- **<resource_name>_id**: id of the impacted record
- **<matchdict value>**: every value matched by each URL pattern name (see Pyramid request matchdict)

And provides the list of affected records in the `impacted_records` attribute. This list contains dictionnaries with `new` and `old` keys. For creation events, only `new` is provided. For deletion events, only `old` is provided. This also allows listeners to react on particular field change or handle *diff* between versions.

Example, when deleting a collection with two records:

```
>>> event.impacted_records
[{'old': {'deleted': True, 'last_modified': 1447240896769, 'id': u'a1f4af60-ddf5-4c49-933f-4cfeff18ac
 {'old': {'deleted': True, 'last_modified': 1447240896770, 'id': u'7a6916aa-0ea1-42a7-9741-c24fe13cb
```

4.7.4 Event listeners

It is possible for an application or a plugin to listen to events and execute some code. Triggered code on events is synchronously called when a request is handled.

Cliquet offers custom listeners that can be activated through configuration, so that every Cliquet-based application can benefit from **pluggable listeners** without using *config.add_event_subscriber()* explicitely.

Currently, a simple built-in listener is available, that just delivers the events into a Redis queue, allowing asynchronous event handling:

class `cliquet.listeners.redis.`**`Listener`**(*client*, *listname*, **args*, ***kwargs*)

A Redis-based event listener that simply pushes the events payloads into the specified Redis list as they happen.

This listener allows actions to be performed asynchronously, using Redis Pub/Sub notifications, or scheduled inspections of the queue.

To activate it, look at *the dedicated configuration*.

Implementing a custom listener consists on implementing the following interface:

class `cliquet.listeners.`**`ListenerBase`**(**args*, ***kwargs*)

`__call__`(*event*)

Parameters **`event`** – Incoming event

4.8 Permissions

Cliquet provides a mechanism to handle authorization on the stored *objects*.

This section gives details about the behaviour of resources in regards to *permissions*.

4.8.1 User resource

This is the simplest one, as presented in the *resource section*.

When using a `cliquet.resource.UserResource`, every authenticated user can manipulate and read their own records. There is no way to restrict this or allow sharing of records.

Method	URL	*permission*
GET / HEAD	/{collection}	*Authenticated*
POST	/{collection}	*Authenticated*
DELETE	/{collection}	*Authenticated*
GET / HEAD	/{collection}/{id}	*Authenticated*
PUT	/{collection}/{id}	*Authenticated*
PATCH	/{collection}/{id}	*Authenticated*
DELETE	/{collection}/{id}	*Authenticated*

Note: When using only these resource, the permission backend remains unused. Its configuration is not necessary.

Public BasicAuth

If *Basic Auth* authentication is enabled, private user resources can become semi-private or public if the `user:pass` is publicly known and shared (for example `public:` is a valid user:pass combination). That's how most simple demos of *Kinto* — a *Cliquet*-based application — are built by the way!

4.8.2 Shareable resource

Warning: When using this kind of resource, the `permission_backend` setting must be set, *as described in the configuration section*.

To introduce more flexibility, the `cliquet.resource.ShareableResource` can be used instead.

```
from cliquet import resource

@resource.register()
class Toadstool(resource.ShareableResource):
    mapping = MushroomSchema()
```

With this alternative resource class, *Cliquet* will register the *endpoints* with a specific route factory, that will take care of checking the appropriate permission for each action.

Method	URL	*permission*	Comments
GET / HEAD	/{collection}	`read`	If not allowed by setting `cliquet.{collection}_read_principals`, will return list of records where user has `read` permission.
POST	/{collection}	`create`	Allowed by setting `cliquet.{collection}_create_principals`
DELETE	/{collection}	`write`	If not allowed by setting `cliquet.{collection}_write_principals`, will delete the list of records where user has `write` permission.
GET / HEAD	/{collection}/{id}	`read`	If not allowed by setting `cliquet.{collection}_read_principals`, will check record permissions
PUT	/{collection}/{id}	`create` if record doesn't exist, `write` otherwise	Allowed by setting `cliquet.{collection}_create_principals`, or `cliquet.{collection}_create_principals` or existing record permissions
PATCH	/{collection}/{id}	`write`	If not allowed by setting `cliquet.{collection}_write_principals`, will check record permissions
DELETE	/{collection}/{id}	`write`	If not allowed by setting `cliquet.{collection}_write_principals`, will check record permissions

The record permissions can be manipulated via the `permissions` attribute in the JSON payload, aside the `data` attribute. It allows to specify the list of *principals* allowed for each `permission`, as detailed in the API section.

Important: When defining permissions, there are two specific principals:

- `system.Authenticated`: any authenticated user
- `system.Everyone`: any user

The `write` permission is required to be able to modify the permissions of an existing record.

When a record is created or modified, the **current user is added to list of principals** for the `write` permission on this object. That means that a user is always able to replace or delete the records she created.

Note: Don't hesitate to submit a contribution to introduce a way to control the current behaviour instead of always granting `write` on current user!

BasicAuth trickery

Like for user resources, if *Basic Auth* authentication is enabled, the predictable user id can be used to define semi-private or public if the `user:pass` is known and shared (for example `public:` is a valid user:pass combination).

For example, get the user id obtained in the hello *root view* with a user:pass combination and use it in the permissions JSON payloads, or settings:

```
cliquet.{collection}_read_principals = basicauth:631c2d625ee5726172cf67c6750de10a3e1a04bcd603bc9ad6d6
```

Related/Inherited permissions

In the above section, the list of allowed principals for actions on the collection (especially `create`) is specified via settings.

It is possible to extend the previously described behavior with related permissions.

For example, in order to imply that having permission to `write` implies permission to `read`. Or having permission to `create` blog articles also means permission to `write` categories.

To do so, specify the `get_bound_permissions` of the *Cliquet* authorization policy.

```python
def get_bound_permissions(self, permission_object_id, permission):
    related = [(permission_object_id, permission)]
    # Grant `read` if user can `write`
    if permission == 'write':
        related.append((permission_object_id, 'read'))
    return related
```

```python
from pyramid.security import IAuthorizationPolicy

def main(global_settings, **settings):
    ...
    cliquet.initialize(config, __version__)
    ...
    authz = config.registry.queryUtility(IAuthorizationPolicy)
    authz.get_bound_permissions = get_bound_permissions
```

In Kinto, this is leveraged to implement an inheritance tree of permissions between nested objects. The root objects permissions still have to be specified via settings though.

It is also possible to subclass the default `cliquet.authorization.AuthorizationPolicy`.

```python
from cliquet import authorization
from pyramid.security import IAuthorizationPolicy
from zope.interface import implementer

@implementer(IAuthorizationPolicy)
class MyAuthz(authorization.AuthorizationPolicy):
    def get_bound_permissions(self, permission_object_id, permission):
        related = [(permission_object_id, permission)]
        # Grant permission on `categories` if permission on `articles`
        if permission_object_id.startswith('/articles'):
            related.append((permission_object_id + '/categories', permission))
        return related
```

This would require forcing the setting `multiauth.authorization_policy = myapp.authz.MyAuthz`.

Manipulate permissions

One way of achieving dynamic permissions is to manipulate the permission backend manually.

For example, in some imaginary admin view:

```python
def admin_view(request):
    # Custom Pyramid view.
    permission = request.registry.permission

    # Give `create` permission to `user_id` in POST
    some_user_id = request.POST['user_id']
    permission_object_id = '/articles'
    permission = 'create'
    permission.add_principal_to_ace(permission_object_id,
                                    permission,
                                    some_user_id)
```

Or during application init (or scripts):

```
def main(global_config, **settings):
    # ...
    cliquet.initialize(config, __version__)
    # ...

    some_user_id = 'basicauth:ut082jghnrgnjnj'
    permission_object_id = '/articles'
    permission = 'create'
    config.registry.permission.add_principal_to_ace(permission_object_id,
                                                    permission,
                                                    some_user_id)
```

Since *principals* can be anything, it is also possible to use them to define groups:

```
def add_to_admins(request):
    # Custom Pyramid view.
    permission = request.registry.permission

    some_user_id = request.POST['user_id']
    group_name = 'group:admins'
    permission.add_user_principal(some_user_id, group_name)
```

And then refer as `group:admins` in the list of allowed principals.

Custom permission checking

The permissions verification in *Cliquet* is done with usual Pyramid authorization abstractions. Most notably using an implementation of a RootFactory in conjonction with an Authorization policy.

In order to completely override (or mimic) the defaults, a custom *RootFactory* and a custom *Authorization policy* can be plugged on the resource during registration.

```
from cliquet import resource

class MyViewSet(resource.ViewSet):

    def get_view_arguments(self, endpoint_type, resource_cls, method):
        args = super(MyViewSet, self).get_view_arguments(endpoint_type,
                                                         resource_cls,
                                                         method)
        if method.lower() not in ('get', 'head'):
            args['permission'] = 'publish'
        return args

    def get_service_arguments(self):
        args = super(MyViewSet, self).get_service_arguments()
        args['factory'] = myapp.MyRootFactory
        return args
```

```
@resource.register(viewset=MyViewSet())
class Resource(resource.UserResource):
    mapping = BookmarkSchema()
```

See more details about available customization in the *viewset section*.

A custom RootFactory and AuthorizationPolicy should implement the permission checking using Pyramid mecanisms.

For example, a simplistic example with the previous resource viewset:

```
from pyramid.security import IAuthorizationPolicy

class MyRootFactory(object):
    def __init__(self, request):
        self.current_resource = None
        service = request.current_service
        if service and hasattr(service, 'resource'):
            self.current_resource = service.resource


@implementer(IAuthorizationPolicy)
class AuthorizationPolicy(object):
    def permits(self, context, principals, permission):
        if context.current_resource == BlogArticle:
            if permission == 'publish':
                return ('group:publishers' in principals)
        return False
```

4.8.3 Backends

The *ACLs* are stored in a *permission* backend. Like for *Storage* and *Cache*, it is pluggable from configuration.

PostgreSQL

class `cliquet.permission.postgresql.`**`Permission`**(*client*, **args*, ***kwargs*)

Permission backend using PostgreSQL.

Enable in configuration:

```
cliquet.permission_backend = cliquet.permission.postgresql
```

Database location URI can be customized:

```
cliquet.permission_url = postgres://user:pass@db.server.lan:5432/dbname
```

Alternatively, username and password could also rely on system user ident or even specified in `~/.pgpass` (*see PostgreSQL documentation*).

Note: Some tables and indices are created when `cliquet migrate` is run. This requires some privileges on the database, or some error will be raised.

Alternatively, the schema can be initialized outside the python application, using the SQL file located in `cliquet/permission/postgresql/schema.sql`. This allows to distinguish schema manipulation privileges from schema usage.

A connection pool is enabled by default:

```
cliquet.permission_pool_size = 10
cliquet.permission_maxoverflow = 10
cliquet.permission_max_backlog = -1
cliquet.permission_pool_recycle = -1
cliquet.permission_pool_timeout = 30
cliquet.cache_poolclass = cliquet.storage.postgresql.pool.QueuePoolWithMaxBacklog
```

The `max_backlog` limits the number of threads that can be in the queue waiting for a connection. Once this limit has been reached, any further attempts to acquire a connection will be rejected immediately, instead of locking up all threads by keeping them waiting in the queue.

See dedicated section in SQLAlchemy documentation for default values and behaviour.

Note: Using a dedicated connection pool is still recommended to allow load balancing, replication or limit the number of connections used in a multi-process deployment.

Noindex

Redis

class `cliquet.permission.redis.`**`Permission`**(*client*, **args*, ***kwargs*)

Permission backend implementation using Redis.

Enable in configuration:

```
cliquet.permission_backend = cliquet.permission.redis
```

(Optional) Instance location URI can be customized:

```
cliquet.permission_url = redis://localhost:6379/2
```

A threaded connection pool is enabled by default:

```
cliquet.permission_pool_size = 50
```

Noindex

Memory

class `cliquet.permission.memory.`**`Permission`**(**args*, ***kwargs*)

Permission backend implementation in local thread memory.

Enable in configuration:

```
cliquet.permission_backend = cliquet.permission.memory
```

Noindex

4.8.4 API

Implementing a custom permission backend consists in implementating the following interface:

class `cliquet.permission.`**`PermissionBase`**(**args*, ***kwargs*)

initialize_schema()

Create every necessary objects (like tables or indices) in the backend.

This is excuted with the `cliquet migrate` command.

flush()

Delete all data stored in the permission backend.

add_user_principal(*user_id*, *principal*)

Add an additional principal to a user.

Parameters

- **user_id** (*str*) – The user_id to add the principal to.
- **principal** (*str*) – The principal to add.

remove_user_principal(*user_id*, *principal*)

Remove an additional principal from a user.

Parameters

- **user_id** (*str*) – The user_id to remove the principal to.
- **principal** (*str*) – The principal to remove.

remove_principal(*principal*)

Remove a principal from every user.

Parameters **principal** (*str*) – The principal to remove.

user_principals(*user_id*)

Return the set of additionnal principals given to a user.

Parameters **user_id** (*str*) – The user_id to get the list of groups for.

Returns The list of group principals the user is in.

Return type *set*

add_principal_to_ace(*object_id*, *permission*, *principal*)

Add a principal to an Access Control Entry.

Parameters

- **object_id** (*str*) – The object to add the permission principal to.
- **permission** (*str*) – The permission to add the principal to.
- **principal** (*str*) – The principal to add to the ACE.

remove_principal_from_ace(*object_id*, *permission*, *principal*)

Remove a principal to an Access Control Entry.

Parameters

- **object_id** (*str*) – The object to remove the permission principal to.
- **permission** (*str*) – The permission that should be removed.
- **principal** (*str*) – The principal to remove to the ACE.

object_permission_principals(*object_id*, *permission*)

Return the set of principals of a bound permission (unbound permission + object id).

Parameters

- **object_id** (*str*) – The object_id the permission is set to.

- **`permission`** (*`str`*) – The permission to query.

Returns The list of user principals

Return type *set*

`principals_accessible_objects`(*principals*, *permission*, *object_id_match=None*, *get_bound_permissions=None*)
Return the list of objects id where the specified *principals* have the specified *permission*.

Parameters

- **`principal`** (*`list`*) – List of user principals
- **`permission`** (*`str`*) – The permission to query.
- **`object_id_match`** (*`str`*) – Filter object ids based on a pattern (e.g. `'*articles*'`).
- **`get_bound_permissions`** (*`function`*) – The methods to call in order to generate the list of permission to verify against. (ie: if you can write, you can read)

Returns The list of object ids

Return type *set*

`object_permission_authorized_principals`(*object_id*, *permission*, *get_bound_permissions=None*)
Return the full set of authorized principals for a given permission + object (bound permission).

Parameters

- **`object_id`** (*`str`*) – The object_id the permission is set to.
- **`permission`** (*`str`*) – The permission to query.
- **`get_bound_permissions`** (*`function`*) – The methods to call in order to generate the list of permission to verify against. (ie: if you can write, you can read)

Returns The list of user principals

Return type *set*

`check_permission`(*object_id*, *permission*, *principals*, *get_bound_permissions=None*)
Test if a principal set have got a permission on an object.

Parameters

- **`object_id`** (*`str`*) – The identifier of the object concerned by the permission.
- **`permission`** (*`str`*) – The permission to test.
- **`principals`** (`set`) – A set of user principals to test the permission against.
- **`get_bound_permissions`** (*`function`*) – The method to call in order to generate the set of permission to verify against. (ie: if you can write, you can read)

`object_permissions`(*object_id*, *permissions=None*)
Return the set of principals for each object permission.

Parameters

- **`object_id`** (*`str`*) – The object_id the permission is set to.
- **`permissions`** (*`list`*) – List of permissions to retrieve. If not define will try to find them all.

Returns The dictionnary with the list of user principals for each object permissions

Return type dict

`replace_object_permissions`(*object_id*, *permissions*)
Replace given object permissions.

Parameters

- **`object_id`** (*`str`*) – The object to replace permissions to.
- **`permissions`** (*`str`*) – The permissions dict to replace.

`delete_object_permissions`(**object_id_list*)
Delete all listed object permissions.

Parameters **`object_id`** (*`str`*) – Remove given objects permissions.

4.9 Errors

`cliquet.errors.`**`http_error`**(*httpexception*, *errno=None*, *code=None*, *error=None*, *message=None*, *info=None*, *details=None*)
Return a JSON formated response matching the error protocol.

Parameters

- **`httpexception`** – Instance of `httpexceptions`
- **`errno`** – stable application-level error number (e.g. 109)
- **`code`** – matches the HTTP status code (e.g 400)
- **`error`** – string description of error type (e.g. "Bad request")
- **`message`** – context information (e.g. "Invalid request parameters")
- **`info`** – information about error (e.g. URL to troubleshooting)
- **`details`** – additional structured details (conflicting record)

Returns the formatted response object

Return type pyramid.httpexceptions.HTTPException

`cliquet.errors.`**`json_error_handler`**(*errors*)
Cornice JSON error handler, returning consistant JSON formatted errors from schema validation errors.

This is meant to be used is custom services in your applications.

```
upload = Service(name="upload", path='/upload',
                 error_handler=errors.json_error_handler)
```

Warning: Only the first error of the list is formatted in the response. (c.f. protocol).

`cliquet.errors.`**`raise_invalid`**(*request*, *location='body'*, *name=None*, *description=None*, ***kwargs*)
Helper to raise a validation error.

Parameters

- **`location`** – location in request (e.g. `'querystring'`)
- **`name`** – field name
- **`description`** – detailed description of validation error

Raises `HTTPBadRequest`

`cliquet.errors.`**`send_alert`**(*request*, *message=None*, *url=None*, *code='soft-eol'*)
Helper to add an Alert header to the response.

Parameters

- **`code`** – The type of error 'soft-eol', 'hard-eol'
- **`message`** – The description message.
- **`url`** – The URL for more information, default to the documentation url.

4.10 Utils

`cliquet.utils.`**`strip_whitespace`**(*v*)
Remove whitespace, newlines, and tabs from the beginning/end of a string.

Parameters **`v`** (*`str`*) – the string to strip.

Return type str

`cliquet.utils.`**`msec_time`**()
Return current epoch time in milliseconds.

Return type int

`cliquet.utils.`**`classname`**(*obj*)
Get a classname from an object.

Return type str

`cliquet.utils.`**`merge_dicts`**(*a*, *b*)
Merge b into a recursively, without overwriting values.

Parameters **`a`** (*`dict`*) – the dict that will be altered with values of *b*.

Return type None

`cliquet.utils.`**`random_bytes_hex`**(*bytes_length*)
Return a hexstring of bytes_length cryptographic-friendly random bytes.

Parameters **`bytes_length`** (*`integer`*) – number of random bytes.

Return type str

`cliquet.utils.`**`native_value`**(*value*)
Convert string value to native python values.

Parameters **`value`** (*`str`*) – value to interprete.

Returns the value coerced to python type

`cliquet.utils.`**`read_env`**(*key*, *value*)
Read the setting key from environment variables.

Parameters

- **`key`** – the setting name
- **`value`** – default value if undefined in environment

Returns the value from environment, coerced to python type `cliquet.utils.`**`encode64`**(*content*, *encoding='utf-8'*)

Encode some content in base64.

Return type str

`cliquet.utils.`**`decode64`**(*encoded_content*, *encoding='utf-8'*)

Decode some base64 encoded content.

Return type str

`cliquet.utils.`**`hmac_digest`**(*secret*, *message*, *encoding='utf-8'*)

Return hex digest of a message HMAC using secret

`cliquet.utils.`**`dict_subset`**(*d*, *keys*)

Return a dict with the specified keys

`cliquet.utils.`**`reapply_cors`**(*request*, *response*)

Reapply cors headers to the new response with regards to the request.

We need to re-apply the CORS checks done by Cornice, in case we're recreating the response from scratch.

`cliquet.utils.`**`current_service`**(*request*)

Return the Cornice service matching the specified request.

Returns the service or None if unmatched.

Return type cornice.Service

`cliquet.utils.`**`current_resource_name`**(*request*)

Return the name used when the Cliquet resource was registered along its viewset.

Returns the resource identifier.

Return type str

`cliquet.utils.`**`build_request`**(*original*, *dict_obj*)

Transform a dict object into a `pyramid.request.Request` object.

It sets a `parent` attribute on the resulting request assigned with the *original* request specified.

Parameters

- **`original`** – the original request.
- **`dict_obj`** – a dict object with the sub-request specifications.

`cliquet.utils.`**`build_response`**(*response*, *request*)

Transform a `pyramid.response.Response` object into a serializable dict.

Parameters

- **`response`** – a response object, returned by Pyramid.
- **`request`** – the request that was used to get the response.

`cliquet.utils.`**`follow_subrequest`**(*request*, *subrequest*, ***kwargs*)

Run a subrequest (e.g. batch), and follow the redirection if any.

Return type tuple

Returns the reponse and the redirection request (or *subrequest* if no redirection happened.)

`cliquet.utils.`**`encode_header`**(*value*, *encoding='utf-8'*)

Make sure the value is of type `str` in both PY2 and PY3.

`cliquet.utils.`**`decode_header`**(*value*, *encoding='utf-8'*)

Make sure the header is an unicode string.

`cliquet.utils.`**`strip_uri_prefix`**(*path*)
    Remove potential version prefix in URI.

class `cliquet.utils.`**`DeprecatedMeta`**
    A metaclass to be set on deprecated classes.

    Warning will happen when class is inherited.

4.11 Glossary

Cliquet Protocol A system of rules that explains the way to interact with the HTTP API *endpoints* (utilities, synchronization, headers etc.), and how data is organized (JSON responses etc.).

CRUD Acronym for Create, Read, Update, Delete

endpoint, endpoints An endpoint handles a particular HTTP verb at a particular URL.

extensible «Extensible» means that the component behaviour can be overriden via lines of code. It differs from «*pluggable*».

Firefox Accounts Account account system run by Mozilla (https://accounts.firefox.com).

HTTP API Multiple publicly exposed endpoints that accept HTTP requests and respond with the requested data, in the form of JSON.

KISS «Keep it simple, stupid» is a design priciple which states that most systems work best if they are kept simple rather than made complicated.

pluggable «Pluggable» means that the component can be replaced via configuration. It differs from «*extensible*».

resource A resource is a collection of records. A resource has two URLs, one for the collection and one for individual records.

user id, user identifier, user identifiers A string that identifies a user. When using the built-in *Basic Auth* authentication, *Cliquet* uses cryptography (HMAC) to generate an identification string.

See Pyramid authentication.

object, objects Also refered as «records», objects are stored by *Cliquet* resources.

tombstone, tombstones When a record is deleted in a resource, a tombstone is created to keep track of the deletion when polling for changes. A tombstone only contains the `id` and `last_modified` fields, everything else is really deleted.

principal, principals An entity that can be authenticated. Principals can be individual people, computers, services, or any group of such things.

permission, permissions An action that can be authorized or denied. read, write, create are permissions.

Semantic Versioning A standard MAJOR.MINOR.PATCH versioning scheme. See http://semver.org/.

ACE, ACEs, Access Control Entity An association of a principal, an object and a permission. For instance, (Alexis, article, write).

ACL, ACLs, Access Control List A list of Access Control Entities (ACE).

CHAPTER 5

Ecosystem

This section gathers information about extending *Cliquet*, and third-party packages.

5.1 Packages

- mozilla-services/cliquet-fxa: Add support of *Firefox Accounts* OAuth2 authentication in *Cliquet*

Note: If you build a package that you would like to see listed here, just get in touch with us!

5.2 Extending Cliquet

5.2.1 Pluggable components

Pluggable components can be substituted from configuration files, as long as the replacement follows the original component API.

```
# myproject.ini
cliquet.logging_renderer = cliquet_fluent.FluentRenderer
```

This is the simplest way to extend *Cliquet*, but will be limited to its existing components (cache, storage, log renderer, ...).

In order to add extra features, including external packages is the way to go!

5.2.2 Include external packages

Appart from usual python «*import and use*», *Pyramid* can include external packages, which can bring views, event listeners etc.

```
import cliquet
from pyramid.config import Configurator


def main(global_config, **settings):
    config = Configurator(settings=settings)
    cliquet.initialize(config, '0.0.1')
    config.scan("myproject.views")

    config.include('cliquet_elasticsearch')

    return config.make_wsgi_app()
```

Alternatively, packages can also be included via configuration:

```
# myproject.ini
cliquet.includes = cliquet_elasticsearch
                   pyramid_debugtoolbar
```

There are many available packages, and it is straightforward to build one.

5.2.3 Include me

In order to be included, a package must define an `includeme(config)` function.

For example, in `cliquet_elasticsearch/init.py`:

```
def includeme(config):
    settings = config.get_settings()

    config.add_view(...)
```

5.2.4 Configuration

In order to ease the management of settings, *Cliquet* provides a helper that reads values from *environment variables* and uses default application values.

```
import cliquet
from pyramid.settings import asbool


DEFAULT_SETTINGS = {
    'cliquet_elasticsearch.refresh_enabled': False
}


def includeme(config):
    cliquet.load_default_settings(config, DEFAULT_SETTINGS)
    settings = config.get_settings()

    refresh_enabled = settings['cliquet_elasticsearch.refresh_enabled']
    if asbool(refresh_enabled):
        ...

    config.add_view(...)
```

In this example, if the environment variable `CLIQUET_ELASTICSEARCH_REFRESH_ENABLED` is set to `true`, the value present in configuration file is ignored.

5.3 Declare API capabilities

Arbitrary capabilities can be declared and exposed in the *root URL*.

Clients can rely on this to detect optional features on the server. For example, features brought by plugins.

```
def main(global_config, **settings):
    config = Configurator(settings=settings)

    cliquet.initialize(config, __version__)
    config.scan("myproject.views")

    settings = config.get_settings()
    if settings['flush_enabled']:
        config.add_api_capability("flush",
                                  description="Flush server using endpoint",
                                  url="http://kinto.readthedocs.io/en/latest/configuration/settings.h

    return config.make_wsgi_app()
```

Note: Any argument passed to `config.add_api_capability()` will be exposed in the root URL.

5.4 Custom backend

As a simple example, let's add add another kind of cache backend to *Cliquet*.

`cliquet_riak/cache.py`:

```
from cliquet.cache import CacheBase
from riak import RiakClient


class Riak(CacheBase):
    def __init__(self, **kwargs):
        self._client = RiakClient(**kwargs)
        self._bucket = self._client.bucket('cache')

    def set(self, key, value, ttl=None):
        key = self._bucket.new(key, data=value)
        key.store()
        if ttl is not None:
            # ...

    def get(self, key):
        fetched = self._bucked.get(key)
        return fetched.data

    #
    # ...see cache documentation for a complete API description.
    #


def load_from_config(config):
    settings = config.get_settings()
    uri = settings['cliquet.cache_url']
    uri = urlparse.urlparse(uri)

    return Riak(pb_port=uri.port or 8087)
```

Once its package installed and available in Python path, this new backend type can be specified in application configuration:

```
# myproject.ini
cliquet.cache_backend = cliquet_riak.cache
```

5.5 Adding features

Another use-case would be to add extra-features, like indexing for example.

- Initialize an indexer on startup;
- Add a `/search/{collection}/` end-point;
- Index records manipulated by resources.

Inclusion and startup in `cliquet_indexing/__init__.py`:

```
DEFAULT_BACKEND = 'cliquet_indexing.elasticsearch'

def includeme(config):
    settings = config.get_settings()
    backend = settings.get('cliquet.indexing_backend', DEFAULT_BACKEND)
    indexer = config.maybe_dotted(backend)

    # Store indexer instance in registry.
    config.registry.indexer = indexer.load_from_config(config)

    # Activate end-points.
    config.scan('cliquet_indexing.views')
```

End-point definitions in `cliquet_indexing/views.py`:

```
from cornice import Service

search = Service(name="search",
                 path='/search/{collection_id}/',
                 description="Search")

@search.post()
def get_search(request):
    collection_id = request.matchdict['collection_id']
    query = request.body

    # Access indexer from views using registry.
    indexer = request.registry.indexer
    results = indexer.search(collection_id, query)

    return results
```

Example indexer class in `cliquet_indexing/elasticsearch.py`:

```
class Indexer(...):
    def __init__(self, hosts):
        self.client = elasticsearch.Elasticsearch(hosts)

    def search(self, collection_id, query, **kwargs):
        try:
            return self.client.search(index=collection_id,
                                      doc_type=collection_id,
                                      body=query,
                                      **kwargs)
        except ElasticsearchException as e:
            logger.error(e)
            raise

    def index_record(self, collection_id, record, id_field):
        record_id = record[id_field]
        try:
            index = self.client.index(index=collection_id,
                                      doc_type=collection_id,
                                      id=record_id,
                                      body=record,
                                      refresh=True)
            return index
        except ElasticsearchException as e:
            logger.error(e)
            raise
```

Indexed resource in `cliquet_indexing/resource.py`:

```
class IndexedModel(cliquet.resource.Model):
    def create_record(self, record):
        r = super(IndexedModel, self).create_record(self, record)

        self.indexer.index_record(self, record)

        return r

class IndexedResource(cliquet.resource.UserResource):
    def __init__(self, request):
        super(IndexedResource, self).__init__(request)
        self.model.indexer = request.registry.indexer
```

Note: In this example, `IndexedResource` must be used explicitly as a base resource class in applications. A nicer pattern would be to trigger *Pyramid* events in *Cliquet* and let packages like this one plug listeners. If you're interested, we started to discuss it!

5.6 JavaScript client

One of the main goal of *Cliquet* is to ease the development of REST microservices, most likely to be used in a JavaScript environment.

A client could look like this:

```
var client = new cliquet.Client({
    server: 'https://api.server.com',
    store: localforage
});
```

```
var articles = client.resource('/articles');

articles.create({title: "Hello world"})
  .then(function (result) {
    // success!
  });

articles.get('id-1234')
  .then(function (record) {
    // Read from local if offline.
  });

articles.filter({
    title: {'$eq': 'Hello'}
  })
  .then(function (results) {
    // List of records.
  });

articles.sync()
  .then(function (result) {
    // Synchronize offline store with server.
  })
  .catch(function (err) {
    // Error happened.
    console.error(err);
  });
```

CHAPTER 6

Troubleshooting

We are doing the best we can so you do not have to read this section.

That said, we have included solutions (or at least explanations) for some common problems below.

If you do not find a solution to your problem here, please *ask for help*!

6.1 ConnectionError: localhost:6379. nodename nor servname provided, or not known

Make sure */etc/hosts* has correct mapping to localhost.

6.2 IOError: [Errno 24] Too many open files

Make sure that max number of connections to redis-server and the max number of file handlers in operating system have access to required memory.

To fix this, increase the open file limit for non-root user:

```
$ ulimit -n 1024
```

6.3 ERROR: InterpreterNotFound: pypy

You need to install Pypy so that it can be found by tox.

CHAPTER 7

Contributing

Thank you for considering to contribute to *Cliquet*!

Note: No contribution is too small; we welcome fixes about typos and grammar bloopers. Don't hesitate to send us a pull request!

Note: Open a pull-request even if your contribution is not ready yet! It can be discussed and improved collaboratively, and avoid having you doing a lot of work without getting feedback.

7.1 Setup your development environment

To prepare your system with Python 3.4, PostgreSQL and Redis, please refer to the *Installation* guide.

You might need to install curl, if you don't have it already.

Prepare your project environment by running:

```
$ make install-dev
```

```
$ pip install tox
```

7.1.1 OS X

On OSX especially you might get the following error when running tests:

```
$ ValueError: unknown locale: UTF-8
```

If this is the case add the following to your ~/.bash_profile:

```
export LC_ALL=en_US.UTF-8
export LANG=en_US.UTF-8
```

Then run:

```
$ source ~/.bash_profile
```

7.2 Run tests

Currently, running the complete test suite implies to run every type of backend.

That means:

- Run Redis on `localhost:6379`
- Run a PostgreSQL 9.4 `testdb` database on `localhost:5432` with user `postgres/postgres`. The database encoding should be `UTF-8`, and the database timezone should be `UTC`.

```
make tests
```

7.2.1 Run a single test

For Test-Driven Development, it is a possible to run a single test case, in order to speed-up the execution:

```
nosetests -s --with-mocha-reporter cliquet.tests.test_views_hello:HelloViewTest.test_returns_info_abo
```

7.2.2 Load tests

A load test is provided in order to run end-to-end tests on *Cliquet* through a sample application, or prevent regressions in terms of performance.

The following `make` command will check briefly the overall sanity of the API, by running a server and running a very few random HTTP requests on it.

```
make loadtest-check-simulation
```

It is possible to customise this load test, by focusing on a single action, or customise the number of users and hits.

First, run the test application manually in a terminal:

```
cd loadtests/
pserve loadtests/testapp.ini
```

And then, run the test suite against it:

```
SERVER_URL=http://localhost:8888 make test -e
```

To run a specific action, specify it with:

```
LOAD_ACTION=batch_create SERVER_URL=http://localhost:8888 make test -e
```

Or a specific configuration:

```
cp test.ini custom.ini
CONFIG=custom.ini SERVER_URL=http://localhost:8888 make test -e
```

7.3 Definition of done

In order to have your changes incorporated, you need to respect these rules:

- **Tests pass**; Travis-CI will build the tests for you on the branch when you push it.

- **Code added comes with tests**; We try to have a 100% coverage on the codebase to avoid surprises. No code should be untested :) If you fail to see how to test your changes, feel welcome to say so in the pull request, we'll gladly help you to find out.
- **Documentation is up to date**;

7.4 IRC channel

If you want to discuss with the team behind *Cliquet*, please come and join us on `#storage` on `irc.mozilla.org`.

- Because of differing time zones, you may not get an immediate response to your question, but please be patient and stay logged into IRC — someone will almost always respond if you wait long enough (it may take a few hours).
- If you don't have an IRC client handy, use the webchat for quick feedback.
- You can direct your IRC client to the channel using this IRC link or you can manually join the #storage IRC channel on the mozilla IRC network.

7.5 How to release

In order to prepare a new release, we are following the following steps.

The *prerelease* and *postrelease* commands are coming from zest.releaser.

Install *zest.releaser* with the *recommended* dependencies. They contain *wheel* and *twine*, which are required to release a new version.

```
$ pip install "zest.releaser[recommended]"
```

7.5.1 Step 1

- Merge remaining pull requests
- Update `CHANGELOG.rst`
- Update version in `cliquet_docs/conf.py`
- Known good versions of dependencies in `requirements.txt`
- Update `CONTRIBUTORS.rst` using: `git shortlog -sne | awk '{$1=""; sub(" ", ""); print}' | awk -F'<' '!x[$1]++' | awk -F'<' '!x[$2]++' | sort`

```
$ git checkout -b prepare-X.Y.Z
$ prerelease
$ vim cliquet_docs/conf.py
$ make build-requirements
$ git commit -a --amend
$ git push origin prepare-X.Y.Z
```

- Open a pull-request with to release the version.

7.5.2 Step 2

Once the pull-request is validated, merge it and do a release. Use the `release` command to invoke the `setup.py`, which builds and uploads to PyPI

```
$ git checkout master
$ git merge --no-ff prepare-X.Y.Z
$ release
$ postrelease
```

7.5.3 Step 3

As a final step:

- Close the milestone in Github
- Add entry in Github release page
- Create next milestone in Github in the case of a major release
- Configure the version in ReadTheDocs
- Send mail to ML (If major release)

That's all folks!

7.6 Cleaning your environment

There are three levels of cleaning your environment:

- `make clean` will remove `*.pyc` files and `__pycache__` directory.
- `make distclean` will also remove `*.egg-info` files and `*.egg`, `build` and `dist` directories.
- `make maintainer-clean` will also remove the `.tox` and the `.venv` directories.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

C

cliquet.errors, 77
cliquet.resource.schema, 53
cliquet.storage, 60
cliquet.storage.exceptions, 63
cliquet.storage.generators, 57
cliquet.utils, 78

Symbols

__call__() (cliquet.listeners.ListenerBase method), 68

A

Access Control Entity, **80**
Access Control List, **80**
ACE, **80**
ACEs, **80**
ACL, **80**
ACLs, **80**
add_principal_to_ace() (cliquet.permission.PermissionBase method), 75
add_user_principal() (cliquet.permission.PermissionBase method), 75
apply_changes() (cliquet.resource.UserResource method), 52
auto_now (cliquet.resource.schema.TimeStamp attribute), 54

B

BackendError, 63
build_request() (in module cliquet.utils), 79
build_response() (in module cliquet.utils), 79

C

Cache (class in cliquet.cache.memory), 65
Cache (class in cliquet.cache.postgresql), 64
Cache (class in cliquet.cache.redis), 65
CacheBase (class in cliquet.cache), 65
check_permission() (cliquet.permission.PermissionBase method), 76
classname() (in module cliquet.utils), 78
Cliquet Protocol, **80**
cliquet.errors (module), 77
cliquet.resource.schema (module), 53
cliquet.storage (module), 60
cliquet.storage.exceptions (module), 63
cliquet.storage.generators (module), 57
cliquet.utils (module), 78
collection (cliquet.resource.UserResource attribute), 50
collection_delete() (cliquet.resource.UserResource method), 51
collection_get() (cliquet.resource.UserResource method), 51
collection_post() (cliquet.resource.UserResource method), 51
collection_timestamp() (cliquet.storage.StorageBase method), 61
create() (cliquet.storage.StorageBase method), 61
create_record() (cliquet.resource.Model method), 55
CRUD, **80**
current_resource_name() (in module cliquet.utils), 79
current_service() (in module cliquet.utils), 79

D

decode64() (in module cliquet.utils), 79
decode_header() (in module cliquet.utils), 79
default_model (cliquet.resource.UserResource attribute), 50
default_viewset (cliquet.resource.UserResource attribute), 50
delete() (cliquet.cache.CacheBase method), 66
delete() (cliquet.resource.UserResource method), 52
delete() (cliquet.storage.StorageBase method), 62
delete_all() (cliquet.storage.StorageBase method), 62
delete_object_permissions() (cliquet.permission.PermissionBase method), 77
delete_record() (cliquet.resource.Model method), 56
delete_records() (cliquet.resource.Model method), 55
deleted_field (cliquet.resource.Model attribute), 54
DeprecatedMeta (class in cliquet.utils), 80
dict_subset() (in module cliquet.utils), 79
direction (cliquet.storage.Sort attribute), 60

E

encode64() (in module cliquet.utils), 78
encode_header() (in module cliquet.utils), 79
endpoint, **80**

endpoints, **80**
expire() (cliquet.cache.CacheBase method), 66
extensible, **80**

F

field (cliquet.storage.Filter attribute), 60
field (cliquet.storage.Sort attribute), 60
Filter (class in cliquet.storage), 60
Firefox Accounts, **80**
flush() (cliquet.cache.CacheBase method), 66
flush() (cliquet.permission.PermissionBase method), 75
flush() (cliquet.storage.StorageBase method), 61
follow_subrequest() (in module cliquet.utils), 79

G

Generator (class in cliquet.storage.generators), 57
get() (cliquet.cache.CacheBase method), 66
get() (cliquet.resource.UserResource method), 51
get() (cliquet.storage.StorageBase method), 61
get_all() (cliquet.storage.StorageBase method), 63
get_name() (cliquet.resource.ViewSet method), 58
get_parent_id() (cliquet.resource.UserResource method), 50
get_record() (cliquet.resource.Model method), 55
get_record_schema() (cliquet.resource.ViewSet method), 58
get_records() (cliquet.resource.Model method), 55
get_service_name() (cliquet.resource.ViewSet method), 58
get_view() (cliquet.resource.ViewSet method), 58
get_view_arguments() (cliquet.resource.ViewSet method), 58

H

hmac_digest() (in module cliquet.utils), 79
HTTP API, **80**
http_error() (in module cliquet.errors), 77

I

id_field (cliquet.resource.Model attribute), 54
initialize() (in module cliquet), 11
initialize_schema() (cliquet.cache.CacheBase method), 65
initialize_schema() (cliquet.permission.PermissionBase method), 74
initialize_schema() (cliquet.storage.StorageBase method), 60
is_endpoint_enabled() (cliquet.resource.ViewSet method), 58
is_known_field() (cliquet.resource.UserResource method), 50
is_readonly() (cliquet.resource.schema.ResourceSchema method), 53

J

json_error_handler() (in module cliquet.errors), 77

K

KISS, **80**

L

Listener (class in cliquet.listeners.redis), 68
ListenerBase (class in cliquet.listeners), 68

M

mapping (cliquet.resource.UserResource attribute), 50
match() (cliquet.storage.generators.Generator method), 57
merge_dicts() (in module cliquet.utils), 78
missing (cliquet.resource.schema.TimeStamp attribute), 54
Model (class in cliquet.resource), 54
modified_field (cliquet.resource.Model attribute), 54
msec_time() (in module cliquet.utils), 78

N

native_value() (in module cliquet.utils), 78

O

object, **80**
object_permission_authorized_principals() (cliquet.permission.PermissionBase method), 76
object_permission_principals() (cliquet.permission.PermissionBase method), 75
object_permissions() (cliquet.permission.PermissionBase method), 76
objects, **80**
operator (cliquet.storage.Filter attribute), 60

P

patch() (cliquet.resource.UserResource method), 51
permission, **80**
Permission (class in cliquet.permission.memory), 74
Permission (class in cliquet.permission.postgresql), 73
Permission (class in cliquet.permission.redis), 74
PermissionBase (class in cliquet.permission), 74
permissions, **80**
PermissionsSchema (class in cliquet.resource.schema), 53
pluggable, **80**
preserve_unknown (cliquet.resource.schema.ResourceSchema.Options attribute), 53
principal, **80**
principals, **80**

principals_accessible_objects() (cliquet.permission.PermissionBase method), 76
process_record() (cliquet.resource.UserResource method), 52
purge_deleted() (cliquet.storage.StorageBase method), 63
put() (cliquet.resource.UserResource method), 51

R

raise_invalid() (in module cliquet.errors), 77
random_bytes_hex() (in module cliquet.utils), 78
read_env() (in module cliquet.utils), 78
readonly_fields (cliquet.resource.schema.ResourceSchema.Options attribute), 53
reapply_cors() (in module cliquet.utils), 79
RecordNotFoundError, 63
regexp (cliquet.storage.generators.Generator attribute), 57
regexp (cliquet.storage.generators.UUID4 attribute), 57
remove_principal() (cliquet.permission.PermissionBase method), 75
remove_principal_from_ace() (cliquet.permission.PermissionBase method), 75
remove_user_principal() (cliquet.permission.PermissionBase method), 75
replace_object_permissions() (cliquet.permission.PermissionBase method), 77
resource, **80**
ResourceSchema (class in cliquet.resource.schema), 53
ResourceSchema.Options (class in cliquet.resource.schema), 53

S

schema_type (cliquet.resource.schema.TimeStamp attribute), 54
schema_type (cliquet.resource.schema.URL attribute), 54
Semantic Versioning, **80**
send_alert() (in module cliquet.errors), 78
set() (cliquet.cache.CacheBase method), 66
Sort (class in cliquet.storage), 60
Storage (class in cliquet.storage.memory), 60
Storage (class in cliquet.storage.postgresql), 59
Storage (class in cliquet.storage.redis), 59
StorageBase (class in cliquet.storage), 60
strip_uri_prefix() (in module cliquet.utils), 79
strip_whitespace() (in module cliquet.utils), 78

T

TimeStamp (class in cliquet.resource.schema), 54
timestamp() (cliquet.resource.Model method), 54
title (cliquet.resource.schema.TimeStamp attribute), 54
tombstone, **80**
tombstones, **80**
ttl() (cliquet.cache.CacheBase method), 66

U

UnicityError, 64
unique_fields (cliquet.resource.schema.ResourceSchema.Options attribute), 53
update() (cliquet.resource.ViewSet method), 58
update() (cliquet.storage.StorageBase method), 61
update_record() (cliquet.resource.Model method), 56
URL (class in cliquet.resource.schema), 54
user id, **80**
user identifier, **80**
user identifiers, **80**
user_principals() (cliquet.permission.PermissionBase method), 75
UserResource (class in cliquet.resource), 50
UUID4 (class in cliquet.storage.generators), 57

V

value (cliquet.storage.Filter attribute), 60
ViewSet (class in cliquet.resource), 58