Cliquet Documentation

Release 2.5.0

Mozilla Services — Da French Team

September 04, 2015

Contents

Cliquet is a toolkit to ease the implementation of HTTP microservices, such as data-driven REST APIs.

- Online documentation
- Issue tracker
- Contributing
- Talk at PyBCN

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

Table of content

1.1 Rationale

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

1.1.1 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.

1.1.2 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.

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.

Generic endpoints

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

Toolkit

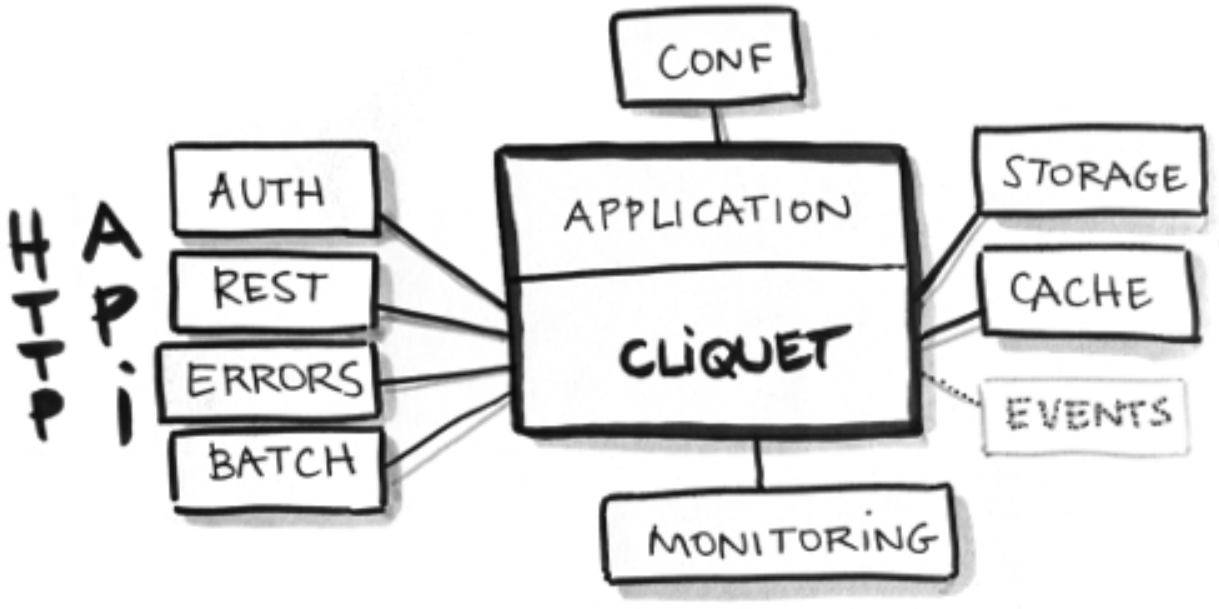


Fig. 1.1: *Cliquet* brings a set of simple but essential features to build APIs.

- 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.

1.1.3 Dependencies

Cliquet is built on the shoulders of giants:

- Cornice for the REST helpers;
- Pyramid for the heavy HTTP stuff;

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

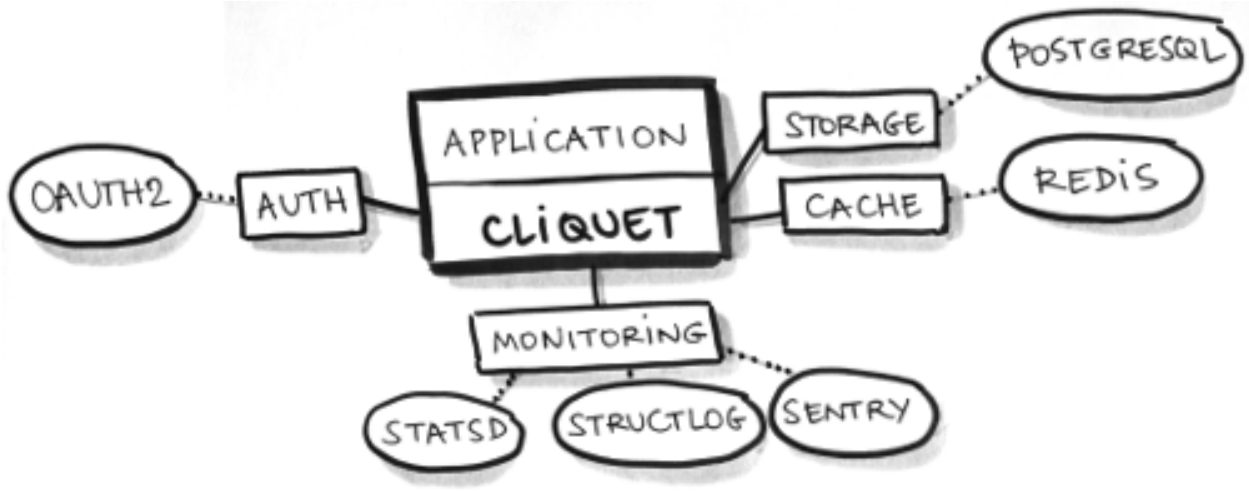


Fig. 1.2: Examples of configuration for a *Cliquet* application in production.

- *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*).

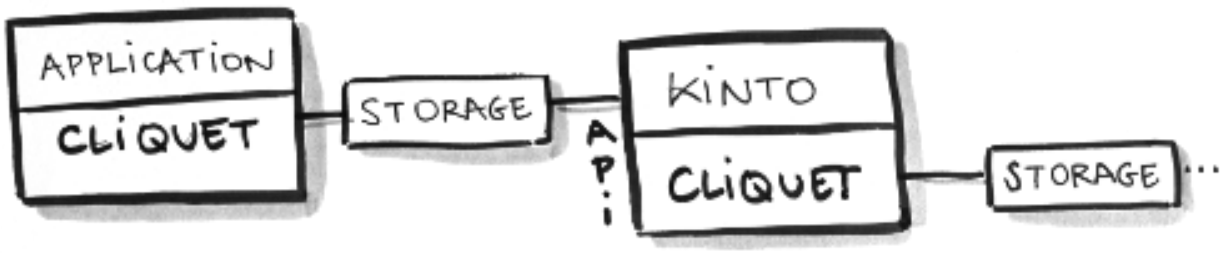
A *Cliquet* application can change or force default values for any setting.

1.1.4 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.
- *Please contact us to add yours*.

Note: Applications built with *Cliquet* can store their data in several kinds of storage backends.
A *Kinto* instance can be used as a storage backend for a *Cliquet* application! *See cloud storage*.



Context

(*to be done*)

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

1.1.5 Vision

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!

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:

- Notifications channel (e.g. run asynchronous tasks on events or listen for changes);
- 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!

1.1.6 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*!

1.2 Getting started

1.2.1 Installation

```
$ pip install cliquet
```

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

1.2.2 Start a Pyramid project

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

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

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 `myproject.ini`.

Note: Shortcut!

In order to bypass the installation and configuration of *Redis* required by the default storage, permission manager and cache, use the «in-memory» backend in `development.ini`:

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

Now is a good time to install the *Cliquet* project locally:

```
$ pip install -e .
```

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 !

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.

1.2.3 Define resources

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

```
from cliquet import resource

@resource.register()
class Mushroom(resource.BaseResource):
```

```
    # No schema yet.
    pass
```

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

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

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

By doing that, a Mushroom resource API is now available at the `/mushrooms/` endpoint.

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.

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.BaseResource):
    mapping = MushroomSchema()
```

1.2.4 What's next ?

Configuration

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

Resource customization

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

Advanced initialization

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.

1.3 HTTP Protocol

1.3.1 API versioning

The API versioning is based on the application version deployed. It follows the semver specifications.

During development the server will be 0.X.X, the server endpoint will be prefixed by `/v0`.

Each non retro-compatible API change will imply the major version number to be incremented. Everything will be made to avoid retro incompatible changes.

The `/` endpoint will redirect to the last API version.

Warning: The version prefix will be **implied** throughout the rest of the API reference, to improve readability. For example, the `/` endpoint should be understood as `/v0/`.

1.3.2 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.

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.

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.

Firefox Accounts

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

1.3.3 Resource endpoints

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;
- `permissions`: *optional* a json dict containing the permissions for the collection of records.

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 attribute 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 attribute with `in_` and provide comma-separated values.

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

Exclude

Prefix attribute name with `not_`:

- `/collection?not_field=0`

Exclude multiple values

Prefix attribute 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 request header `If-None-Match` 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*).

Changed in version 2.4:::It will be supported until the next major version of Cliquet.

Request:

_to was renamed _before and is now deprecated.

```
GET /buckets/default/collections/tasks/records?_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.

List of available URL parameters

- `<prefix?><attribute name>`: filter by value(s)
- `_since`, `_before`: polling changes
- `_sort`: order list
- `_limit`: pagination max size
- `_token`: pagination token

Filtering, sorting and paginating can all be combined together.

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

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
- `412 Precondition Failed`: Collection changed since value in `If-Match` header

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 request header `If-Match` is provided, and if the record has changed meanwhile, 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 (per user), 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 **:ref:'dedicated section about errors <error-responses>'_**.

HTTP Status Codes

- `201 Created`: The record was created
- `400 Bad Request`: The request body is invalid
- `409 Conflict`: Unicity constraint on fields is violated
- `412 Precondition Failed`: Collection changed since value in `If-Match` header

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 request header `If-Match` 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;
- `405 Method Not Allowed`: This endpoint is not available;
- `412 Precondition Failed`: Collection changed since value in `If-Match` header

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 request header `If-None-Match` 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
- `412 Precondition Failed`: Record changed since value in `If-Match` header

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 request header `If-Match` 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.

HTTP Status Code

- `200 OK`: The record was deleted
- `412 Precondition Failed`: Record changed since value in `If-Match` header

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.

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 request header `If-Match` is provided, and if the record has changed meanwhile, 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"
    }
}
```

HTTP Status Code

- `201 Created`: The record was created
- `200 OK`: The record was replaced
- `400 Bad Request`: The record is invalid
- `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.

Note: A `If-None-Match: *` request header can be used to make sure the `PUT` won't overwrite any record.

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;
- `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 newly created permissions dict, containing the permissions for the modified record.

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.

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 request header `If-Match` is provided, 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.

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*).

HTTP Status Code

- `200 OK`: The record was modified
- `400 Bad Request`: The request body is invalid, or a read-only field was modified
- `409 Conflict`: If modifying this record violates a field unicity constraint
- `412 Precondition Failed`: Record changed since value in `If-Match` header

Protected resources

All of the described endpoints can be either *protected* or not. Protecting and enpoint means that only *principals* which have been granted access will be able to issue requests successfully.

In the case of a *protected* resource, body is a JSON mapping containing a `permissions` key in addition to the `data` key. Permissions can also be replaced and modified independantly from data.

On a request, `permissions` is a json dict containing the permissions for the record to be modified. It has the following signature:

```
'permissions': {'{permission}': [{list_of_principals}]}
```

{permission} is a placeholder for the permission name (e.g. *read*, *write*, *create*) and *{list_of_principals}* should be replaced by an actual list of principals.

`permissions` is also added to JSON mapping response bodies, and contains the *modified* version of the permissions in case of a modification, or the list of permissions in case of a read operation.

1.3.4 Batch operations

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

```
{
  "defaults": {
    "method" : "POST",
    "path" : "/v0/articles",
    "headers" : {
      ...
    }
  },
  "requests": [
    {
      "body" : {
        "title": "MoFo",
        "url" : "http://mozilla.org",
        "added_by": "FxOS",
      }
    },
    {
      "body" : {
        "title": "MoCo",
        "url" : "http://mozilla.com"
```

```
        "added_by": "FxOS",
      }
    },
    {
      "method" : "PATCH",
      "path" : "/articles/409",
      "body" : {
        "read_position" : 3477
      }
    }
  ]
}
```

The response body is a list of all responses:

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

HTTP Status Codes

- `200 OK`: The request has been processed
- `400 Bad Request`: The request body is invalid

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

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

Pros & Cons

- This respects REST principles
- This is easy for the client to handle, since it just has to pile up HTTP requests while offline
- It looks to be a convention for several REST APIs (Neo4J, Facebook, Parse)
- Payload of response can be heavy, especially while importing huge collections
- Payload of response must all be iterated to look-up errors

Note: A form of payload optimization for massive operations is planned.

1.3.5 Utility endpoints for OPS and Devs

GET /

The returned value is a JSON mapping containing:

- `hello`: the name of the service (e.g. `"reading list"`)
- `version`: complete version (`"X.Y.Z"`)
- `commit`: the HEAD git revision number when run from a git repository.
- `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)
- `documentation`: The URL to the service documentation. (this document!)
- **`settings`: a mapping with the values of relevant public settings for clients**
  - `cliquet.batch_max_requests`: Number of requests that can be made in a batch request.
- `userid`: The connected perso user id. The field is not present when no Authorization header is provided.

GET /__heartbeat__

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

- `storage` true if operational
- `cache` true if operational
- `oauth` true if operational, or *null* if not enabled

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

1.3.6 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.

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 if still good.

Concurrency control

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

The client can then choose to:

- overwrite by repeating the request without `If-Match`;
- reconcile the resource by fetching, merging and repeating the request.

1.3.7 Backoff indicators

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.

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
```

1.3.8 Error responses

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>*.

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"
```

```
        }
    },
}
```

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"
        }
    }
}
```

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"
        }
    ]
}
```

1.3.9 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.

1.4 Internals

1.4.1 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.

Supported Python versions

Cliquet supports Python 2.7, Python 3.4 and PyPy.

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.

Python 3.4

Linux

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

OS X

```
brew install python3.4
```

Cryptography libraries

Linux

On Debian / Ubuntu based systems:

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

On RHEL-derivatives:

```
apt-get install libffi-devel openssl-devel
```

OS X

Assuming brew is installed:

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

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
```

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
```

1.4.2 Configuration

See Pyramid settings documentation.

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.

Project info

```
cliquet.project_name = project
cliquet.project_docs = https://project.rtfd.org/
# cliquet.project_version = 1.0
```

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
```

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
```

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
```

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.

Cache

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

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

See *cache backend documentation* for more details.

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 **:github:'pyramid multiauth <mozilla-service/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.

Permission configuration

ACE 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`.

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
```

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)
```

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_json_serializer
                                  cliquet.initialization.setup_logging
                                  cliquet.initialization.setup_storage
                                  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_stats
```

1.4.3 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.

Full example

```
import colander

from cliquet import resource
from cliquet import schema
from cliquet import utils


class BookmarkSchema(resource.ResourceSchema):
    url = schema.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.BaseResource):
    mapping = BookmarkSchema()

    def process_record(self, new, old=None):
        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.

Resource 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']))
```

Resource class

In order to customize the resource URLs or behaviour on record processing, the resource class can be extended:

Interaction with storage

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

```
from cliquet import resource


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


class Payment(resource.BaseResource):
    def __init__(request):
        super(Payment, self).__init__(request)
        self.collection = TrackedCollection(
            storage=self.collection.storage,
            id_generator=self.collection.id_generator,
            collection_id=self.collection.collection_id,
            parent_id=self.collection.parent_id,
            auth=self.collection.auth)
```

Custom record ids

By default, records ids are *UUID4 <http://en.wikipedia.org/wiki/Universally_unique_identifier>_*.

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.BaseResource):
    def __init__(request):
        super(Mushroom, self).__init__(request)
        self.collection.id_generator = MsecId()
```

Generators objects

Custom Usage

Within views

In 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.Collection(storage=registry.storage,
                                  name='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 remote *Kinto* collection into a local storage:

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


config_local = Configurator(settings=DEFAULT_SETTINGS)
config_local.add_settings({
    'cliquet.storage_backend': 'cliquet.storage.postgresql'
    'cliquet.storage_url': 'postgres://user:pass@db.server.lan:5432/dbname'
})
local = resource.Collection(storage=config_local.registry.storage,
                            parent_id='browsing',
                            name='history')

config_remote = Configurator(settings=DEFAULT_SETTINGS)
config_remote.add_settings({
    'cliquet.storage_backend': 'kinto.storage',
    'cliquet.storage_url': 'https://cloud-storage.services.mozilla.com'
})
remote = resource.Collection(storage=config_remote.registry.storage,
                             parent_id='browsing',
                             name='history',
                             auth='Basic bWF0Og==')

records, total = in remote.get_records():
for record in records:
    local.create_record(record)
```

1.4.4 Viewsets

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

View sets can be viewed as a set of rules which can be applied to a resource in order to define what should be inserted in the routing mechanism of pyramid.

Configuring a viewset

To use *Cliquet* in a basic fashion, there is not need to understand how viewsets work in full detail, but it might be useful to know how to extend the defaults.

Default viewset can be extended by passing viewset arguments to the *resource.register* class decorator:

```
from cliquet import resource


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

Subclassing a viewset

In case this isn't enough to update the default properties, the default *ViewSet* class can be subclassed in a more specific viewset, and then be passed during the registration phase:

```
from cliquet import resource


class MyViewSet(resource.ViewSet):

    def get_service_name(self, endpoint_type, resource):
        """Returns the name of the service, depending a given type and
        resource.
        """
        # Get the resource name from an akwards location.
        return name


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

ViewSet class

In order to customize the resource URLs or permissions, the viewset class can be extended:

1.4.5 Storage

Backends

PostgreSQL

Redis

Memory

Cloud Storage

Note: Under construction

If the `kinto` package is available, it is possible to store data in a remote instance of *Kinto*.

```
cliquet.storage_backend = kinto.storage
cliquet.storage_url = https://cloud-storage.services.mozilla.com
```

See Kinto for more details.

Note: In order to avoid double checking of OAuth tokens, the Kinto service and the application can share the same cache (`cliquet.cache_url`).

API

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

Exceptions

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 collection class to manipulate collections of records.

1.4.6 Cache

PostgreSQL

Redis

Memory

API

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

1.4.7 Permissions

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

Glossary

Authorization isn't complicated, but requires the introduction of a few terms so that explanations are easier to follow:

Object: The data that is stored into *Cliquet*. *objects* usually match the resources you defined; For one resource there are two *objects*: resource's collection and resource's records.

Principal: An entity that can be authenticated. *principals* can be individual people, computers, services, or any group of such things.

Permission: An action that can be authorized or denied. *read*, *write*, *create* are *permissions*.

Access Control Entity (ACE): An association of a *principal*, an *object* and a *permission*. For instance, (Alexis, article, write).

Access Control List (ACL): A list of Access Control Entities (*ACE*).

Overview

By default, the resources defined by *Cliquet* are public, and records are isolated by user. But it is also possible to define *protected resources*, which will required the user to have access to the requested resource.

```
from cliquet import authorization
from cliquet import resource


@resource.register(factory=authorization.RouteFactory)
class Toadstool(resource.ProtectedResource):
    mapping = MushroomSchema()
```

In this example, a *route factory* is registered. Route factories are explained in more details below.

A protected resource, in addition to the `data` property of request / responses, takes a *permissions* property which contains the list of *principals* that are allowed to access or modify the current *object*.

During the creation of the *object*, the *permissions* property is stored in the permission backend, and upon access, it checks the current *principal* has access the the object, with the correct permission.

Route factory

The route factory decides which *permission* is required to access one resource or another. Here is a summary of the *permissions* that are defined by the default route factory *Cliquet* defines:

Method	*permission*
POST	`create`
GET / HEAD	`read`
PUT	`create` if it doesn't exist, `write` otherwise
PATCH	`write`
DELETE	`write`

Route factories are best described in the pyramid documentation

Authorization policy

Upon access, the authorization policy is asked if any of the current list of *principals* has access to the current resource. By default, the authorization policy *Cliquet* checks in the *permission* backend for the current *object*.

It is possible to extend this behavior, for instance if there is an inheritance tree between the defined resources (some *ACEs* should give access to its child *objects*).

In case the application should define its own inheritance tree, it should also define its own authorization policy.

To do so, subclass the default `AuthorizationPolicy` and add a specific `get_bound_permission` method.

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

@implementer(IAuthorizationPolicy)
class AuthorizationPolicy(authorization.AuthorizationPolicy):
    def get_bound_permissions(self, *args, **kwargs):
    """Callable that takes an object ID and a permission and returns
    a list of tuples (<object id>, <permission>)."""
        return build_permissions_set(*args, **kwargs)
```

Permissions backend

The *ACLs* are stored in a *permission* backend. Currently, permission backends exists for Redis and PostgreSQL, as well as a in memory one. It is of course possible to add you own permission backend, if you whish to store your *permissions* related data in a different database.

1.4.8 Errors

1.4.9 Utils

1.4.10 Glossary

CRUD Acronym for Create, Read, Update, Delete

endpoint 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).

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.

user id, user identifier, user identifiers A string that identifies a user. By default, *Cliquet* uses a HMAC on authentication credentials to generate users identifications strings.

See Pyramid authentication.

object, objects The data that is stored into *Cliquet*. Objects usually match the resources you defined; For one resource there are two objects: resource's collection and resource's records.

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.

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).

1.5 Ecosystem

This section gathers information about extending *Cliquet*, and third-party packages.

1.5.1 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!

1.5.2 Extending Cliquet

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!

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
pyramid.includes = cliquet_elasticsearch
                   pyramid_debugtoolbar
```

There are **'many available packages'_**, and it is straightforward to build one.

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(...)
```

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.

1.5.3 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
```

1.5.4 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 IndexedCollection(cliquet.resource.Collection):
    def create_record(self, record):
        r = super(IndexedCollection, self).create_record(self, record)

        self.indexer.index_record(self, record)

        return r

class IndexedResource(cliquet.resource.BaseResource):
    def __init__(self, request):
        super(IndexedResource, self).__init__(request)
        self.collection.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!

1.5.5 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);
  });
```

1.6 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.

1.6.1 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
```

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
```

1.6.2 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
```

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
```

1.6.3 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**;

1.6.4 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.

1.6.5 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]
```

Step1

- Merge remaining pull requests
- Update changelog
- Update version in docs/conf.py

- Version dependencies + Remove master url in dev-requirements.txt
- If cliquet was updated, update the link to default settings in the docs

```
$ git checkout -b prepare-X.X
$ prerelease
$ vim docs/conf.py
$ rm -rf .venv
$ make install && .venv/bin/pip freeze > requirements.txt
$ # Remove the master URL in the requirements.txt file.
$ git commit -a --amend
$ git push origin prepare-X.X
```

Step 2

Once the pull request is validated, you can merge it and do a release. We are using the *release* command to invoke the *setup.py* builds and upload to PyPI.

```
$ git checkout master
$ git merge --no-ff prepare-X.X
$ release
$ postrelease
```

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!

1.7 Changelog

This document describes changes between each past release.

1.7.1 2.5.0 (2015-09-04)

Protocol

- Collection records can now be filtered using multiple values (`?in_status=1,2,3`) (fixes #39)
- Collection records can now be filtered excluding multiple values (`?exclude_status=1,2,3`) (fixes mozilla-services/readinglist#68)

Internal changes

- We can obtains accessible objects_id in a collection from user principals (fixes #423)

1.7.2 2.4.3 (2015-08-26)

Bug fixes

- Fix the packaging for cliquet (#430)

1.7.3 2.4.2 (2015-08-26)

Internal changes

- Remove the symlink to cliquet_docs and put the documentation inside *cliquet_docs* directly (#426)

1.7.4 2.4.1 (2015-08-25)

Internal changes

- Make documentation available from outside by using *cliquet_docs* (#413)

1.7.5 2.4.0 (2015-08-14)

Protocol

- Userid is now provided when requesting the hello endpoint with an `Authorization` header (#319)
- UUID validation now accepts any kind of UUID, not just v4 (fixes #387)
- Querystring parameter `_to` was renamed to `_before` (*the former is now deprecated*) (#391)

New features

- Cliquet `Service` class now has the default error handler attached (#388)
- Allow to configure info link in error responses with `cliquet.error_info_link` setting (#395)
- Storage backend now has a `purge_deleted()` to get rid of tombstones (#400)

Bug fixes

- Fix missing `Backoff` header for 304 responses (fixes #416)
- Fix Python3 encoding errors (#328)
- `data` is not mandatory in request body if the resource does not define any schema or if no field is mandatory (fixes mozilla-services/kinto#63)
- Fix no validation error on PATCH with unknown attribute (fixes #374)
- Fix permissions not validated on PATCH (fixes #375)
- Fix CORS header missing in 404 responses for unknown URLs (fixes #414)

Internal changes

- Renamed main documentation sections to *HTTP Protocol* and *Internals* (#394)
- Remove mentions of storage in documentation to avoid confusions with the *Kinto* project.
- Add details in timestamp documentation.
- Mention talk at Python Meetup Barcelona in README
- Fix documentation about postgres-contrib dependancy (#409)

- Add `cliquet.utils` to *Internals* documentation (#407)
- Default id generator now accepts dashes and underscores (#411)

1.7.6 2.3.1 (2015-07-15)

Bug fixes

- Fix crash on hello view when application is not deployed from Git repository (fixes #382)
- Expose Content-Length header to Kinto.js (#390)

1.7.7 2.3 (2015-07-13)

New features

- Provide details about existing record in `412` error responses (fixes mozilla-services/kinto#122)
- Add ETag on record PUT/PATCH responses (fixes #352)
- Add StatsD counters for the permission backend

Bug fixes

- Fix crashes in permission backends when permission set is empty (fixes #368, #371)
- Fix value of ETag on record: provide collection timestamp on collection endpoints only (fixes #356)
- Default resources do accept `permissions` attribute in payload anymore
- Default resources do not require a root factory (fixes #348)
- Default resources do not hit the permission backend anymore
- Default viewset was split and does not handle permissions anymore (fixes #322)
- Permissions on views is now set only on resources
- Fix missing `last_modified` field in PATCH response when no field was changed (fixes #371)
- Fix lost querystring during version redirection (fixes #364)

Internal changes

- Document the list of public settings in hello view (mozilla-services/kinto#133)

1.7.8 2.2.1 (2015-07-06)

Bug fixes

- Fix permissions handling on PATCH /resource (#358)

1.7.9 2.2.0 (2015-07-02)

New features

- Add public settings in hello view (#318)

Bug fixes

- Fix version redirection behaviour for unsupported versions (#341)

- PostgreSQL dependencies are now fully optional in code (#340)
- Prevent overriding final settings from `default_settings` parameter in `cliquet.initialize()` (#343)

Internal changes

- Fix installation documentation regarding PostgreSQL 9.4 (#338, thanks @elemoine!)
- Add detail about UTC and UTF-8 for PostgreSQL (#347, thanks @elemoine!)
- Remove UserWarning exception when running tests (#339, thanks @elemoine!)
- Move build_request and build_response to `cliquet.utils` (#344)
- Pypy is now tested on Travis CI (#337)

1.7.10 2.1.0 (2015-06-26)

New features

- Cliquet does not require authentication policies to prefix user ids anymore (fixes #299).
- Pypy support (thanks Balthazar Rouberol #325)
- Allow to override parent id of resources (#333)

Bug fixes

- Fix crash in authorization on `OPTIONS` requests (#331)
- Fix crash when `If-Match` is provided without `If-None-Match` (#335)

Internal changes

- Fix docstrings and documentation (#329)

1.7.11 2.0.0 (2015-06-16)

New features

- Authentication and authorization policies, as well as group finder function can now be specified via configuration (fixes #40, #265)
- Resources can now be protected by fine-grained permissions (#288 via #291, #302)

Minor

- Preserve provided `id` field of records using POST on collection (#293 via #294)
- Logging value for authentication type is now available for any kind of authentication policy.
- Any resource endpoint can now be disabled from settings (#46 via #268)

Bug fixes

- Do not limit cache values to string (#279)
- When PUT creates the record, the HTTP status code is now 201 (#298, #300)
- Add safety check in `utils.current_service()` (#316)

Breaking changes

- `cliquet.storage.postgresql` now requires PostgreSQL version 9.4, since it now relies on *JSONB*. Data will be migrated automatically using the `migrate` command.

- the `@crud` decorator was replaced by `@register()` (fixes #12, #268)
- Firefox Accounts code was removed and published as external package *cliquet-fxa*
- The *Cloud storage* storage backend was removed out of *Cliquet* and should be revamped in *Kinto* repository (mozilla-services/kinto#45)

API

- Resource endpoints now expect payloads to have a `data` attribute (#254, #287)
- Resource endpoints switched from `If-Modified-Since` and `If-Unmodified-Since` to `Etags` (fixes #251 via #275), thanks @michielbdejong!

Minor

- `existing` attribute of conflict errors responses was moved inside a generic `details` attribute that is also used to list validation errors.
- Setting `cliquet.basic_auth_enabled` is now deprecated. Use pyramid_multiauth configuration instead to specify authentication policies.
- Logging value for authentication type is now `authn_type` (with `FxAOAuth` or `BasicAuth` as default values).

Internal changes

- Cliquet resource code was split into `Collection` and `Resource` (fixes #243, #282)
- Cleaner separation of concern between `Resource` and the new notion of `ViewSet` (#268)
- Quickstart documentation improvement (#271, #312) thanks @N1k0 and @brouberol!
- API versioning documentation improvements (#313)
- Contribution documentation improvement (#306)

1.7.12 1.8.0 (2015-05-13)

Breaking changes

- Switch PostgreSQL storage to JSONB: requires 9.4+ (#104)
- Resource name is not a Python property anymore (ref #243)
- Return existing record instead of raising 409 on POST (fixes #75)
- `cliquet.storage.postgresql` now requires version PostgreSQL 9.4, since it now relies on *JSONB*. Data will be migrated automatically using the `migrate` command.
- Conflict errors responses `existing` attribute was moved inside a generic `details` attribute that is also used to list validation errors.
- In heartbeat end-point response, `database` attribute was renamed to `storage`

New features

- Storage records ids are now managed in python (fixes #71, #208)
- Add setting to disable version redirection (#107, thanks @hiromipaw)
- Add response behaviour headers for PATCH on record (#234)
- Provide details in error responses (#233)
- Expose new function `cliquet.load_default_settings()` to ease reading of settings from defaults and environment (#264)

- Heartbeat callback functions can now be registered during startup (#261)

Bug fixes

- Fix migration behaviour when metadata table is flushed (#221)
- Fix backoff header presence if disabled in settings (#238)

Internal changes

- Require 100% of coverage for tests to pass
- Add original error message to storage backend error
- A lots of improvements in documentation (#212, #225, #228, #229, #237, #246, #247, #248, #256, #266, thanks Michiel De Jong)
- Migrate *Kinto* storage schema on startup (#218)
- Fields `id` and `last_modified` are not part of resource schema anymore (#217, mozilla-services/readinlist#170)
- Got rid of redundant indices in storage schema (#208, ref #138)
- Disable Cornice schema request binding (#172)
- Do not hide FxA errors (fixes mozilla-services/readinglist#70)
- Move initialization functions to dedicated module (ref #137)
- Got rid of request custom attributes for storage and cache (#245)

1.7.13 1.7.0 (2015-04-10)

Breaking changes

- A **command must be ran during deployment** for database schema migration:

  $ cliquet –ini production.ini migrate

- Sentry custom code was removed. Sentry logging is now managed through the logging configuration, as explained in docs.

New features

- Add PostgreSQL schema migration system (#139)
- Add cache and oauth in heartbeat view (#184)
- Add monitoring features using NewRelic (#189)
- Add profiling features using Werkzeug (#196)
- Add ability to override default settings in initialization (#136)
- Add more statsd counter for views and authentication (#200)
- Add in-memory cache class (#127)

Bug fixes

- Fix crash in DELETE on collection with PostgreSQL backend
- Fix Heka logging format of objects (#199)
- Fix performance of record insertion using ordered index (#138)
- Fix 405 errors not JSON formatted (#88)

- Fix basic auth prompt when disabled (#182)

Internal changes

- Improve development setup documentation (thanks @hiromipaw)
- Deprecated `cliquet.initialize_cliquet`, renamed to `cliquet.initialize`.
- Code coverage of tests is now 100%
- Skip unstable tests on TravisCI, caused by `fsync = off` in their PostgreSQL.
- Perform random creation and deletion in heartbeat view (#202)

1.7.14 1.6.0 (2015-03-30)

New features

- Split schema initialization from application startup, using a command-line tool.

```
cliquet --ini production.ini init
```

Bug fixes

- Fix connection pool no being shared between cache and storage (#176)
- Default connection pool size to 10 (instead of 50) (#176)
- Warn if PostgreSQL session has not UTC timezone (#177)

Internal changes

- Deprecated `cliquet.storage_pool_maxconn` and `cliquet.cache_pool_maxconn` settings (renamed to `cliquet.storage_pool_size` and `cliquet.cache_pool_size`)

1.7.15 1.5.0 (2015-03-27)

New features

- Mesure calls on the authentication policy (#167)

Breaking changes

- Prefix statsd metrics with the value of *cliquet.statsd_prefix* or *cliquet.project_name* (#162)
- *http_scheme* setting has been replaced by *cliquet.http_scheme* and *cliquet.http_host* was introduced ((#151, #166)
- URL in the hello view now has version prefix (#165)

Bug fixes

- Fix Next-Page url if service has key in url (#158)
- Fix some PostgreSQL connection bottlenecks (#170)

Internal changes

- Update of PyFxA to get it working with gevent monkey patching (#168)
- Reload kinto on changes (#158)

1.7.16 1.4.1 (2015-03-25)

Bug fixes

- Rely on Pyramid API to build pagination Next-Url (#147)

1.7.17 1.4.0 (2015-03-24)

Breaking changes

- Make monitoring dependencies optional (#121)

Bug fixes

- Force PostgreSQl session timezone to UTC (#122)
- Fix basic auth ofuscation and prefix (#128)
- Make sure the *paginate_by* setting overrides the passed *limit* argument (#129)
- Fix limit comparison under Python3 (#143)
- Do not serialize using JSON if not necessary (#131)
- Fix crash of classic logger with unicode (#142)
- Fix crash of CloudStorage backend when remote returns 500 (#142)
- Fix behaviour of CloudStorage with backslashes in querystring (#142)
- Fix python3.4 segmentation fault (#142)
- Add missing port in Next-Page header (#147)

Internal changes

- Use ujson again, it was removed in the 1.3.2 release (#132)
- Add index for as_epoch(last_modified) (#130). Please add the following statements to SQL for the migration:

```
ALTER FUNCTION as_epoch(TIMESTAMP) IMMUTABLE;
CREATE INDEX idx_records_last_modified_epoch ON records(as_epoch(last_modified));
CREATE INDEX idx_deleted_last_modified_epoch ON deleted(as_epoch(last_modified));
```

- Prevent fetching to many records for one user collection (#130)
- Use UPSERT for the heartbeat (#141)
- Add missing OpenSSL in installation docs (#146)
- Improve tests of basic auth (#128)

1.7.18 1.3.2 (2015-03-20)

- Revert ujson usage (#132)

1.7.19 1.3.1 (2015-03-20)

Bug fixes

- Fix packaging (#118)

1.7.20 1.3.0 (2015-03-20)

New features

- Add PostgreSQL connection pooling, with new settings `cliquet.storage_pool_maxconn` and `cliquet.cache_pool_maxconn` (*Default: 50*) (#112)
- Add StatsD support, enabled with `cliquet.statsd_url = udp://server:port` (#114)
- Add Sentry support, enabled with `cliquet.sentry_url = http://user:pass@server/1` (#110)

Bug fixes

- Fix FxA verification cache not being used (#103)
- Fix heartbeat database check (#109)
- Fix PATCH endpoint crash if request has no body (#115)

Internal changes

- Switch to ujson for JSON de/serialization optimizations (#108)

1.7.21 1.2.1 (2015-03-18)

- Fix tests about unicode characters in BATCH querystring patch
- Remove CREATE CAST for the postgresql backend
- Fix environment variable override

1.7.22 1.2 (2015-03-18)

Breaking changes

- *cliquet.storage.postgresql* now uses UUID as record primary key (#70)
- Settings `cliquet.session_backend` and `cliquet.session_url` were renamed `cliquet.cache_backend` and `cliquet.cache_url` respectively.
- FxA user ids are not hashed anymore (#82)
- Setting `cliquet.retry_after` was renamed `cliquet.retry_after_seconds`
- OAuth2 redirect url now requires to be listed in `fxa-oauth.webapp.authorized_domains` (e.g. `*.mozilla.com`)
- Batch are now limited to 25 requests by default (#90)

New features

- Every setting can be specified via an environment variable (e.g. `cliquet.storage_url` with `CLIQUET_STORAGE_URL`)
- Logging now relies on structlog (#78)
- Logging output can be configured to stream JSON (#78)
- New cache backend for PostgreSQL (#44)
- Documentation was improved on various aspects (#64, #86)
- Handle every backend errors and return 503 errors (#21)
- State verification for OAuth2 dance now expires after 1 hour (#83)

Bug fixes

- FxA OAuth views errors are now JSON formatted (#67)
- Prevent error when pagination token has bad format (#72)
- List of CORS exposed headers were fixed in POST on collection (#54)

Internal changes

- Added a method in *cliquet.resource.Resource* to override known fields (*required by Kinto*)
- Every setting has a default value
- Every end-point requires authentication by default
- Session backend was renamed to cache (#96)

1.7.23 1.1.4 (2015-03-03)

- Update deleted_field support for postgres (#62)

1.7.24 1.1.3 (2015-03-03)

- Fix include_deleted code for the redis backend (#60)
- Improve the update_record API (#61)

1.7.25 1.1.2 (2015-03-03)

- Fix packaging to include .sql files.

1.7.26 1.1.1 (2015-03-03)

- Fix packaging to include .sql files.

1.7.27 1.1 (2015-03-03)

New features

- Support filter on deleted using since (#51)

Internal changes

- Remove python 2.6 support (#50)
- Renamed Resource.deleted_mark to Resource.deleted_field (#51)
- Improve native_value (#56)
- Fixed Schema options inheritance (#55)
- Re-build the virtualenv when setup.py changes
- Renamed storage.url to cliquet.storage_url (#49)
- Refactored the tests/support.py file (#38)

1.7.28 1.0 (2015-03-02)

- Initial version, extracted from Mozilla Services Reading List project (#1)

New features

- Expose CORS headers so that client behind CORS policy can access them (#5)
- Postgresql Backend (#8)
- Use RedisSession as a cache backend for PyFxA (#10)
- Delete multiple records via DELETE on the collection_path (#13)
- Batch default prefix for endpoints (#14 / #16)
- Use the app version in the / endpoint (#22)
- Promote Basic Auth as a proper authentication backend (#37)

Internal changes

- Backends documentation (#15)
- Namedtuple for filters and sort (#17)
- Multiple DELETE in Postgresql (#18)
- Improve Resource API (#29)
- Refactoring of error management (#41)
- Default Options for Schema (#47)

1.8 Contributors

- Alexis Metaireau <alexis@mozilla.com>
- Andy McKay <amckay@mozilla.com>
- Balthazar Rouberol <br@imap.cc>
- Dan Phrawzty <phrawzty+github@gmail.com>
- Éric Lemoine <eric.lemoine@gmail.com>
- Hiromipaw <silvia@nopressure.co.uk>
- Mathieu Leplatre <mathieu@mozilla.com>
- Michiel de Jong <michiel@unhosted.org>
- Nicolas Perriault <nperriault@mozilla.com>
- Rémy Hubscher <rhubscher@mozilla.com>
- Tarek Ziade <tarek@mozilla.com>

CHAPTER 2

Indices and tables

- genindex
- modindex
- search