py-couchdb Documentation

Release 1.12

Andrey Antukh

Feb 20, 2020

Contents

Release v1.12.

py-couchdb is a BSD Licensed, modern pure Python CouchDB client.

Currently there are several libraries for Python to connect to CouchDB. **Why one more?** It's very simple. All seem to be not maintained, all libraries use standard Python libraries for http requests, and are not compatible with Python3.

CHAPTER 1

Advantages of py-couchdb

- Uses requests for http requests (much faster than the standard library)
- Python2 and Python3 compatible with same codebase (with one exception, Python view server that uses 2to3)
- Also compatible with pypy.

Note: requests 1.2 seems buggy with Python3 and I strongly recommend use request 1.1 if you use Python3

Example:

```
>>> import pycouchdb
>>> server = pycouchdb.Server("http://admin:admin@localhost:5984/")
>>> server.info()['version']
'1.2.1'
```

CHAPTER 2

User guide

This part of the documentation gives a simple introduction on py-couchdb usage.

2.1 Installation

This part of the documentation covers the installation of `py-couchdb`.

2.1.1 Distribute & Pip

Installing `py-couchdb` is simple with pip:

```
$ pip install pycouchdb
```

2.1.2 Cheeseshop Mirror

If the Cheeseshop is down, you can also install Requests from one of the mirrors. Crate.io is one of them:

```
$ pip install -i http://simple.crate.io/ pycouchdb
```

2.1.3 Get the Code

`py-couchdb` is actively developed on GitHub, where the code is always available.

You can either clone the public repository:

```
git clone git://github.com/histrio/py-couchdb.git
```

Once you have a copy of the source, you can embed it in your Python package, or install it into your site-packages easily:

```
$ python setup.py install
```

2.2 Quickstart

This page gives a good introduction in how to get started with py-couchdb. This assumes you already have it installed. If you do not, head over to the *Installation* section.

2.2.1 Connect to a server

Connect to a couchdb server is very simple. Begin by importing `pycouchdb` module and instance a server class:

```
>>> import pycouchdb
>>> server = pycouchdb.Server()
```

2.2.2 Authentication

By default, py-couchdb connects to a `http://localhost:5984/` but if your couchdb requieres authentication, you can pass `http://username:password@localhost:5984/` to server constructor:

```
>>> server = pycouchdb.Server("http://username:password@localhost:5984/")
```

py-couchdb have two methods for authentication: with session or basic auth. By default, "session" method is used but if you like, can specify the method on create a server instance:

```
>>> server = pycouchdb.Server("http://username:password@localhost:5984/",
                                                        authmethod="basic")
```

2.2.3 Create, obtain and delete a database

CouchDB can contains multiple databases. For access to one, this is a simple example:

```
>>> db = server.database("foo")
>>> db
<pycouchdb.client.Database object at 0x7fd4ae835dd0>
```

Can create one new db:

```
>>> server.create("foo2")
<pycouchdb.client.Database object at 0x7f9c46059310>
```

And can remove a database:

```
>>> server.delete("foo2")
```

If you intent remove not existent database, *NotFound* exception is raised. For more information see Exceptions API.

```
>>> server.delete("foo")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
```

(continues on next page)

(continued from previous page)

```
  File "./pycouchdb/client.py", line 42, in delete
    raise NotFound("database {0} not found".format(name))
pycouchdb.exceptions.NotFound: database foo not found
```

2.2.4 Create, obtain and delete a document

The simplest way for get a document is using its `id`.

```
>>> db = server.database("foo")
>>> doc = db.get("b1536a237d8d14f3bfde54b41b036d8a")
>>> doc
{'_rev': '1-d62e11770282e4fcc5f6311dae6c80ee', 'name': 'Bar',
                    '_id': 'b1536a237d8d14f3bfde54b41b036d8a'}
```

You can create an own document:

```
>>> doc = db.save({"name": "FOO"})
>>> doc
{'_rev': '1-6a1be826ddbd67649df8aa1e0bf12da1',
'_id': 'ef9e608db6434dd39ab3dc4cf35d22b7', 'name': 'FOO'}
```

Delete a document:

```
>>> db.delete("ef9e608db6434dd39ab3dc4cf35d22b7")
>>> "ef9e608db6434dd39ab3dc4cf35d22b7" not in db
True
```

2.2.5 Querying a database

With couchDB you can make two types of queries: temporary or view. This is a simple way to make a temporary query:

```
>>> map_func = "function(doc) { emit(doc.name, 1); }"
>>> db.temporary_query(map_func)
<generator object _query at 0x7f65bd292870>
>>> list(db.temporary_query(map_func))
[{'value': 1, 'id': '8b588fa0a3b74a299c6d958467994b9a', 'key': 'Fooo'}]
```

And this is a way to make a query using predefined views:

```
>>> _doc = {
...     "_id": "_design/testing",
...     "views": {
...         "names": {
...             "map": "function(doc) { emit(doc.name, 1); }",
...             "reduce": "function(k, v) { return  sum(v); }",
...         }
...     }
...}
>>> doc = db.save(_doc)
>>> list(db.query("testing/names", group='true'))
[{'value': 1, 'key': 'Fooo'}]
```

In order to make query with Python see *Views* on how to configure CouchDB. And this is a way to make a query using predefined views with Python:

```
>>> _doc = {
...     "_id": "_design/testing",
...     "language": "python3",
...     "views": {
...         "names": {
...             "map": "def fun(doc): yield doc.name, 1",
...             "reduce": "def fun(k, v): return  sum(v)",
...         }
...     }
...}
>>> doc = db.save(_doc)
>>> list(db.query("testing/names", group='true', language='python3'))
[{'value': 1, 'key': 'Fooo'}]
```

2.2.6 Subscribe to a changes stream feed

CouchDB exposes a fantastic stream API for push change notifications, and with **pycouchdb** you can subscribe to these changes in a very simple way:

```
>>> def feed_reader(message, db):
...     print(message)
...
>>> db.changes_feed(feed_reader)
```

`changes_feed` blocks until a stream is closed or `FeedReaderExited` is raised inside of reader function.

Also, you can make reader as class. This have some advantage, because it exposes often useful close callback.

Example:

```
>>> from pycouchdb.feedreader import BaseFeedReader
>>> from pycouchdb.exceptions import FeedReaderExited
>>>
>>> class MyReader(BaseFeedReader):
...     def on_message(self, message):
...         # self.db is a current Database instance
...         # process message
...         raise FeedReaderExited()
...
...     def on_close(self):
...         # This is executed after a exception
...         # is raised on on_message method
...         print("Feed reader end")
...
>>> db.changes_feed(MyReader())
```

2.3 Views in python

The pycouchdb package comes with a view server to allow you to write views in Python instead of JavaScript. When pycouchdb is installed, it will install a script called couchpy that runs the view server. To enable this for your CouchDB server, add the following section to local.ini:

```
[query_servers]
python3 = /path/to/couchpy
```

After restarting CouchDB, the Futon view editor should show **python3** in the language pull-down menu. Here's some sample view code to get you started:

```
def fun(doc):
    if "name" in doc:
        yield doc['name'], None
```

Note: The view server also works with python 2.7 and pypy.

2.4 Developer Interface

This part of documentation covers a main developer interface. All py-couchdb functionality can be accessed by these classes:

2.4.1 Server

class pycouchdb.client.**Server**(*base_url='http://localhost:5984/'*, *full_commit=True*, *authmethod='basic'*, *verify=False*)

Class that represents a couchdb connection.

Parameters

- **verify** – setup ssl verification.
- **base_url** – a full url to couchdb (can contain auth data).
- **full_commit** – If `False`, couchdb not commits all data on a request is finished.
- **authmethod** – specify a authentication method. By default "basic" method is used but also exists "session" (that requires some server configuration changes).

changes_feed(*feed_reader*, ***kwargs*)

Subscribe to changes feed of the whole CouchDB server.

Note: this method is blocking.

Parameters **feed_reader** – callable or `BaseFeedReader` instance

config()

Get a current config data.

create(*name*)

Create a database.

Parameters **name** – database name

Raises `Conflict` if a database already exists

Returns a *`Database`* instance

database(*name*)

Get a database instance.

Parameters **name** – database name

Raises `NotFound` if a database does not exists

Returns a *Database* instance

delete(*name*)
Delete some database.

Parameters **name** – database name

Raises `NotFound` if a database does not exists

info()
Get server info.

Returns dict with all data that couchdb returns.

Return type dict

replicate(*source*, *target*, ***kwargs*)
Replicate the source database to the target one.

New in version 1.3.

Parameters

- **source** – URL to the source database
- **target** – URL to the target database

stats(*name=None*)
Get runtime stats.

Parameters **name** – if is not None, get stats identified by a name.

Returns dict

version()
Get the current version of a couchdb server.

2.4.2 Database

class `pycouchdb.client.`**Database**(*resource*, *name*)
Class that represents a couchdb database.

all(*wrapper=None*, *flat=None*, *as_list=False*, ***kwargs*)
Execute a builtin view for get all documents.

Parameters

- **wrapper** – wrap result into a specific class.
- **as_list** – return a list of results instead of a default lazy generator.
- **flat** – get a specific field from a object instead of a complete object.

Returns generator object

changes_feed(*feed_reader*, ***kwargs*)
Subscribe to changes feed of couchdb database.

Note: this method is blocking.

Parameters **feed_reader** – callable or `BaseFeedReader` instance

changes_list(***kwargs*)
Obtain a list of changes from couchdb.

cleanup()
Execute a cleanup operation.

commit()
Send commit message to server.

compact()
Send compact message to server. Compacting write-heavy databases should be avoided, otherwise the process may not catch up with the writes. Read load has no effect.

compact_view(*ddoc*)
Execute compact over design view.

Raises `NotFound` if a view does not exists.

config()
Get database status data such as document count, update sequence etc. :return: dict

delete(*doc_or_id*)
Delete document by id.

Changed in version 1.2: Accept document or id.

Parameters **doc_or_id** – document or id

Raises `NotFound` if a document not exists

Raises `Conflict` if delete with wrong revision.

delete_attachment(*doc*, *filename*)
Delete attachment by filename from document.

Changed in version 1.2: Now returns a new document instead of modify the original.

Parameters

- **doc** – document dict
- **filename** – name of attachment.

Raises `Conflict` if save with wrong revision.

Returns doc

delete_bulk(*docs*, *transaction=True*)
Delete a bulk of documents.

New in version 1.2.

Parameters **docs** – list of docs

Raises `Conflict` if a delete is not success

Returns raw results from server

get(*doc_id*, *params=None*, ***kwargs*)
Get a document by id.

Parameters **doc_id** – document id

Raises `NotFound` if a document not exists

Returns document (dict)

get_attachment(*doc*, *filename*, *stream=False*, ***kwargs*)
Get attachment by filename from document.

Parameters

- **doc** – document dict
- **filename** – attachment file name.
- **stream** – setup streaming output (default: False)

Returns binary data or

one(*name*, *flat=None*, *wrapper=None*, ***kwargs*)
Execute a design document view query and returns a firts result.

Parameters

- **name** – name of the view (eg: docidname/viewname).
- **wrapper** – wrap result into a specific class.
- **flat** – get a specific field from a object instead of a complete object.

Returns object or None

put_attachment(*doc*, *content*, *filename=None*, *content_type=None*)
Put a attachment to a document.

Changed in version 1.2: Now returns a new document instead of modify the original.

Parameters

- **doc** – document dict.
- **content** – the content to upload, either a file-like object or bytes
- **filename** – the name of the attachment file; if omitted, this function tries to get the filename from the file-like object passed as the *content* argument value

Raises `Conflict` if save with wrong revision.

Raises ValueError

Returns doc

query(*name*, *wrapper=None*, *flat=None*, *as_list=False*, ***kwargs*)
Execute a design document view query.

Parameters

- **name** – name of the view (eg: docidname/viewname).
- **wrapper** – wrap result into a specific class.
- **as_list** – return a list of results instead of a default lazy generator.
- **flat** – get a specific field from a object instead of a complete object.

Returns generator object

revisions(*doc_id*, *status='available'*, *params=None*, ***kwargs*)
Get all revisions of one document.

Parameters

- **doc_id** – document id
- **status** – filter of reverion status, set empty to list all

Raises `NotFound` if a view does not exists.

Returns generator object

save(*doc*, *batch=False*)
Save or update a document.

Changed in version 1.2: Now returns a new document instead of modify the original.

Parameters

- **doc** – document
- **batch** – allow batch=ok inserts (default False)

Raises `Conflict` if save with wrong revision.

Returns doc

save_bulk(*docs*, *transaction=True*)
Save a bulk of documents.

Changed in version 1.2: Now returns a new document list instead of modify the original.

Parameters

- **docs** – list of docs
- **transaction** – if `True`, couchdb do a insert in transaction model.

Returns docs

temporary_query(*map_func*, *reduce_func=None*, *language='javascript'*, *wrapper=None*, *as_list=False*, ***kwargs*)
Execute a temporary view.

Parameters

- **map_func** – unicode string with a map function definition.
- **reduce_func** – unicode string with a reduce function definition.
- **language** – language used for define above functions.
- **wrapper** – wrap result into a specific class.
- **as_list** – return a list of results instead of a default lazy generator.
- **flat** – get a specific field from a object instead of a complete object.

Returns generator object

Bibliography

[Ref] http://docs.couchdb.org/en/1.6.1/api/server/common.html#db-updates

Index