pyramid_redis_sessions Documentation

Release 1.0.1

Eric Rasmussen

December 21, 2015

Contents

This package provides a fast and stable implementation of Pyramid's ISession interface, using Redis as its backend.

Special thanks to Chris McDonough for the original idea, inspiration, and some borrowed code.

CHAPTER 1

When to Use Redis for Sessions

Persistent session stores are ideal when you want server side sessions and a clean separation of concerns (your code doesn't need to know details as long as it knows how to talk to the server). Redis expands on these benefits by giving us:

- built-in key expiration to automatically clean up expired session data
- no need for complicated/unpredictable lock handling in our python code
- a lightweight alternative to full transactions (the watch mechanism)

CHAPTER 2

When Not to Use Redis for Sessions

Redis makes a compelling case for session data, but as with any technology decision it's important to be aware of the trade-offs. Adding Redis to your stack can mean:

- time spent installing, configuring, and maintaining a Redis instance
- speed before consistency (Redis is fast at the cost of syncing *eventually*)
- the entirety of your session data must fit in memory

Typically these aren't concerns for sessions, because critical data doesn't usually belong in a client session. However, in specialized cases where you need consistency at the cost of speed, you may consider database-backed sessions using a proven database server like PostgreSQL.

Alternatively, if you only ever store less than ~4kb of non-sensitive data, cookie-based sessions work nicely without requiring you to add complexity to your stack.

CHAPTER 3

Narrative Documentation

3.1 Getting Started

3.1.1 Installation

It is recommended that you add pyramid_redis_sessions to your pyramid app's setup.py file so that it will be automatically installed and managed. For instance, in the *setup* call in your setup.py you can add *pyramid_redis_sessions* to the *requires* list:

```
requires = [
    'pyramid',
    'pyramid_redis_sessions',
]
setup(
  # all your package metadata
  install_requires = requires,
)
```

But for a quick start, you can also get the package from PyPI with either:

```
$ easy_install pyramid_redis_sessions
```

Or if you prefer:

```
$ pip install pyramid_redis_sessions
```

For Redis installation notes see Redis Notes.

3.1.2 Configuration

Next, configure *pyramid_redis_sessions* via your Paste config file. Only *redis.sessions.secret* is required. All other settings are optional.

For complete documentation on the *RedisSessionFactory* that uses these settings, see API. Otherwise, keep reading for the quick list:

```
# session settings
redis.sessions.secret = your_cookie_signing_secret
redis.sessions.timeout = 1200

# session cookie settings
redis.sessions.cookie_name = session
```

```
redis.sessions.cookie_max_age = max_age_in_seconds
redis.sessions.cookie_path = /
redis.sessions.cookie_domain =
redis.sessions.cookie_secure = False
redis.sessions.cookie_httponly = False
redis.sessions.cookie_on_exception = True

# you can supply a redis connection string as a URL
redis.sessions.url = redis://username:password@localhost:6379/0

# or as individual settings (note: the URL gets preference if you do both)
redis.sessions.host = localhost
redis.sessions.port = 6379
redis.sessions.db = 0
redis.sessions.password = None

# additional options can be supplied to redis-py's StrictRedis
redis.sessions.socket_timeout =
redis.sessions.connection_pool =
redis.sessions.charset = utf-8
redis.sessions.errors = strict
redis.sessions.unix_socket_path =

# in the advanced section we'll cover how to instantiate your own client
redis.sessions.client_callable = my.dotted.python.callable

# along with defining your own serialize and deserialize methods
redis.sessions.serialize = cPickle.dumps
redis.sessions.deserialize = cPickle.loads

# you can specify a prefix to be used with session keys in redis
redis.sessions.prefix = mycoolprefix

# or you can supply your own UID generator callable for session keys
redis.sessions.id_generator = niftyuid
```

3.1.3 Initialization

Lastly, you need to tell Pyramid to use *pyramid_redis_sessions* as your session factory. The preferred way is adding it with *config.include*, like this:

```
def main(global_config, **settings):
    config = Configurator(settings=settings)
    config.include('pyramid_redis_sessions')
```

Alternately, instead of using the Configurator's `include` method, you can activate Pyramid by changing your application's `.ini` file, use the following line:

```
pyramid.includes = pyramid_redis_sessions
```

The above method is recommended because it's simpler, idiomatic, and still fully configurable. It even has the added benefit of automatically resolving dotted python paths used in the advanced options (see Advanced Usage).

However, you can also explicitly pass a *settings* dict to the *session_factory_from_settings* function. This can be helpful if you configure or modify your settings in code:

```
from pyramid_redis_sessions import session_factory_from_settings

def main(global_config, **settings):
    config = Configurator(settings=settings)
    session_factory = session_factory_from_settings(settings)
    config.set_session_factory(session_factory)
```

3.2 Advanced Usage

3.2.1 Adjusting Timeouts Dynamically

It's useful to think of a session as a way to manage online loitering. If you had a brick and mortar store, you wouldn't want people sitting around for hours at a time not shopping. The session timeout is the physical world equivalent of some tough looking security folk that politely escort loiterers from the building.

But one day the loiterers might be the store owners, or your grandparents, or people you don't want thrown out after a couple of minutes of not shopping. In the physical world you'd need to spend time training the security team to treat those people specially. In *pyramid_redis_sessions*, you only need to identify one of these users and call the following method:

```
request.session.adjust_timeout_for_session(timeout_in_seconds)
```

This will permanently change the timeout setting for that user's session for the duration of the session.

3.2.2 Supplying Your Own Redis Client

pyramid_redis_sessions makes things easy for most developers by creating a Redis client from settings and storing the client in Pyramid's *registry* for later use. However, you may find yourself wanting extra control over how the client is created.

Here are some reasons you might want to build your own client callable:

- you want to use your own wrapper or redis-py's Redis instead of StrictRedis
- you want to choose from multiple Redis instances or modify connection settings based on the current request

To this or other ends, you can specify a dotted python path to a custom Redis client callable:

```
redis.sessions.client_callable = app.module.my_connection_getter
```

If you instantiate the session factory with includeme, Pyramid's *config* machinery will follow the dotted path and attempt to return the callable.

However, if you instantiate the session factory in code (even by passing in a settings dict), you must supply the actual python callable rather than a dotted string.

Either way, the python object must be a callable that takes a Pyramid request and the keyword arguments accepted by StrictRedis (you don't *have* to use StrictRedis, but those are the Redis-specific settings that will be passed to your callable).

Example:

```
def get_redis_client(request, **redis_options):
    redis = get_redis_instance_from_somewhere()
    if not redis:
        redis = StrictRedis(**redis_options)
```

```
        set_redis_instance_somewhere(redis)
    return redis
```

Special thanks to raydeo on #pyramid for the idea.

3.2.3 Overriding cPickle

By default, *pyramid_redis_sessions* uses *cPickle* for serializing and deserializing sessions to and from Redis. *cPickle* is very fast, stable, and widely used, so I recommend sticking with it unless you have a specific reason not to.

However, because you may very well have a specific reason not to, you can specify the following settings in your config:

```
redis.sessions.serialize = my_module.my_serializer
redis.sessions.deserialize = my_module.my_deserializer
```

If you do change the defaults you're on your own, and it's assumed that the following holds:

```
decode(encode(data)) == data
```

Where *data* is, at minimum, a python dict of session data.

One possible use case (given that redis does not support encryption or decryption) is supplying an encode function that serializes then encrypts, and a decode function that decrypts then deserializes. However, there will be a performance penalty for encrypting and decrypting all session data all the time. If you only need to encrypt some sensitive data, a simpler solution would be adding the encrypted data to the session and decrypting it when you retrieve it from the session.

3.2.4 Overriding the id_generator

pyramid_redis_sessions has a sensible and recommended default for quickly generating unique session keys. You don't need to specify anything to use it.

However, if you'd like to prefix the keys (typically for visual inspection in redis) you can use:

```
redis.sessions.prefix = mycoolprefix
```

And if for any reason you want to generate unique IDs on your own or using a particular UID function, you can specify a callable with:

```
redis.sessions.id_generator = some_library.some_uid_generating_function
```

This is useful when you want to increase security at the cost of performance, reduce integrity for greater speed on a small internal app, or any other specialized tradeoff. But again, unless you have highly specialized requirements, please use the default.

3.3 API

pyramid_redis_sessions.**RedisSessionFactory**(*secret*, *timeout=1200*, *cookie_name='session'*, *cookie_max_age=None*, *cookie_path='/'*, *cookie_domain=None*, *cookie_secure=False*, *cookie_httponly=True*, *cookie_on_exception=True*, *url=None*, *host='localhost'*, *port=6379*, *db=0*, *password=None*, *socket_timeout=None*, *connection_pool=None*, *charset='utf-8'*, *errors='strict'*, *unix_socket_path=None*, *client_callable=None*, *serialize=<built-in function dumps>*, *deserialize=<built-in function loads>*, *id_generator=<function _generate_session_id>*)

Constructs and returns a session factory that will provide session data from a Redis server. The returned factory can be supplied as the `session_factory` argument of a `pyramid.config.Configurator` constructor, or used as the `session_factory` argument of the `pyramid.config.Configurator.set_session_factory()` method.

Parameters:

`secret` A string which is used to sign the cookie.

`timeout` A number of seconds of inactivity before a session times out.

`cookie_name` The name of the cookie used for sessioning. Default: `session`.

`cookie_max_age` The maximum age of the cookie used for sessioning (in seconds). Default: `None` (browser scope).

`cookie_path` The path used for the session cookie. Default: `/`.

`cookie_domain` The domain used for the session cookie. Default: `None` (no domain).

`cookie_secure` The 'secure' flag of the session cookie. Default: `False`.

`cookie_httponly` The 'httpOnly' flag of the session cookie. Default: `True`.

`cookie_on_exception` If `True`, set a session cookie even if an exception occurs while rendering a view. Default: `True`.

`url` A connection string for a Redis server, in the format: redis://username:password@localhost:6379/0 Default: `None`.

`host` A string representing the IP of your Redis server. Default: `localhost`.

`port` An integer representing the port of your Redis server. Default: `6379`.

`db` An integer to select a specific database on your Redis server. Default: `0`

`password` A string password to connect to your Redis server/database if required. Default: `None`.

`client_callable` A python callable that accepts a Pyramid *request* and Redis config options and returns a Redis client such as redis-py's *StrictRedis*. Default: `None`.

`serialize` A function to serialize the session dict for storage in Redis. Default: `cPickle.dumps`.

`deserialize` A function to deserialize the stored session data in Redis. Default: `cPickle.loads`.

`id_generator` A function to create a unique ID to be used as the session key when a session is first created. Default: private function that uses sha1 with the time and random elements to create a 40 character unique ID.

The following arguments are also passed straight to the `StrictRedis` constructor and allow you to further configure the Redis client:

```
socket_timeout
connection_pool
charset
errors
unix_socket_path
```

pyramid_redis_sessions.**includeme**(*config*)

This function is detected by Pyramid so that you can easily include *pyramid_redis_sessions* in your *main* method like so:

```
config.include('pyramid_redis_sessions')
```

Parameters:

`config` A Pyramid `config.Configurator`

pyramid_redis_sessions.**session_factory_from_settings**(*settings*)

Convenience method to construct a `RedisSessionFactory` from Paste config settings. Only settings prefixed with "redis.sessions" will be inspected and, if needed, coerced to their appropriate types (for example, casting the `timeout` value as an *int*).

Parameters:

`settings` A dict of Pyramid application settings

RedisSession.**adjust_timeout_for_session**(*session*, **arg*, ***kw*)

3.4 Redis Notes

3.4.1 Installing Redis

The best place to start is the Redis quick start guide.

If you need automated deployment with your application, you can find guides online for Redis deployment via buildout, puppet, chef, etc. If anyone would like to compile a list of recipes for these deployment options, I wholly encourage pull requests with links.

Discussions of Redis security are outside the purview of these docs, but it's worth noting that Redis will listen on all interfaces by default, potentially exposing your data to the world. You can avoid this with a bind declaration in your redis.conf file such as:

```
bind 127.0.0.1
```

You can read more in a blog post discussing this issue here.

3.4.2 Why Redis?

Redis is fast, widely deployed, and stable. It works best when your data can fit in memory, but is configurable and still quite fast when you need to sync to disk. There are plenty of existing benchmarks, opinion pieces, and articles if you want to learn about its use cases. But for *pyramid_redis_sessions*, I'm interested in it specifically for these reasons:

- it really is bleeping fast (choose your own expletive)
- it has a very handy built-in mechanism for setting expirations on keys

- the watch mechanism is a nice, lightweight alternative to full transactions
- session data tends to be important but not mission critical, but if it is...
- it has configurable persistence

3.5 Contributing

3.5.1 Feature Additions/Requests

I'm very interested in discussing use cases that *pyramid_redis_sessions* doesn't cover but that you'd like to see in your session library.

If you have an idea you want to discuss further, ping me (erasmas) on freenode in #pyramid, or you're also welcome to submit a pull request.

However, I do ask that you make the request on a new feature.<your feature> branch so that I can spend some time testing the code before merging to master.

3.5.2 Notes on Testing

The test suite is written in a way that may be unusual to some, so if you submit a patch I only ask that you follow the testing methodology employed here. On a technical level it boils down to:

1. Parameterizing classes or functions that connect to outside systems
2. In tests, supplying dummy instances of those classes

In practice this means never hardcoding a redis-py *StrictRedis* instance in *pyramid_redis_sessions*, and always passing in instances of *DummyRedis* in tests.

On a philosophical level I see outside processes as swappable strategies, and the purpose of my code is to control how those strategies are employed. For this reason tests in *pyramid_redis_session* should never need to use *Mock*.

CHAPTER 4

Support and Documentation

The official documentation is available at: http://pyramid-redis-sessions.readthedocs.org/en/latest/index.html

You can report bugs or open support requests in the github issue tracker, or you can discuss issues with me (erasmas) and other users in #pyramid on irc.freenode.org.

CHAPTER 5

Authors

Eric Rasmussen is the primary author, but owes much to Chris McDonough and the fine folks from the Pyramid community. A complete list of contributors is available in CONTRIBUTORS.txt.

CHAPTER 6

License

pyramid_redis_sessions is available under a FreeBSD-derived license. See LICENSE.txt for details.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

p

A

I

P

R

S