Redis Timeseries Documentation

Release 0.1.8

Ryan Anguiano

Jul 26, 2017

Contents

Contents:

CHAPTER 1

Redis Timeseries

Time series API built on top of Redis that can be used to store and query time series statistics. Multiple time granularities can be used to keep track of different time intervals.

- Free software: MIT license
- Documentation: https://redis-timeseries.readthedocs.io.

Install

To install Redis Timeseries, run this command in your terminal:

```
$ pip install redis_timeseries
```

Usage

To initialize the TimeSeries class, you must pass a Redis client to access the database. You may also override the base key for the time series.

```
>>> import redis
>>> client = redis.StrictRedis()
>>> ts = TimeSeries(client, base_key='my_timeseries')
```

To customize the granularities, make sure each granularity has a `ttl` and `duration` in seconds. You can use the helper functions for easier definitions.

```
>>> my_granularities = {
...     '1minute': {'ttl': hours(1), 'duration': minutes(1)},
...     '1hour': {'ttl': days(7), 'duration': hours(1)}
... }
>>> ts = TimeSeries(client, granularities=my_granularities)
```

`.record_hit()` accepts a key and an optional timestamp and increment count. It will record the data in all defined granularities.

```
>>> ts.record_hit('event:123')
>>> ts.record_hit('event:123', datetime(2017, 1, 1, 13, 5))
>>> ts.record_hit('event:123', count=5)
```

`.record_hit()` will automatically execute when `execute=True`. If you set `execute=False`, you can chain the commands into a single redis pipeline. You must then execute the pipeline with `.execute()`.

```
>>> ts.record_hit('event:123', execute=False)
>>> ts.record_hit('enter:123', execute=False)
>>> ts.record_hit('exit:123', execute=False)
>>> ts.execute()
```

`.get_hits()` will query the database for the latest data in the selected granularity. If you want to query the last 3 minutes, you would query the `1minute` granularity with a count of 3. This will return a list of tuples `(bucket, count)` where the bucket is the rounded timestamp.

```
>>> ts.get_hits('event:123', '1minute', 3)
[(datetime(2017, 1, 1, 13, 5), 1), (datetime(2017, 1, 1, 13, 6), 0), (datetime(2017,␣
↪1, 1, 13, 7), 3)]
```

`.get_total_hits()` will query the database and return only a sum of all the buckets in the query.

```
>>> ts.get_total_hits('event:123', '1minute', 3)
4
```

`.scan_keys()` will return a list of keys that could exist in the selected range. You can pass a search string to limit the keys returned. The search string should always have a `*` to define the wildcard.

```
>>> ts.scan_keys('1minute', 10, 'event:*')
['event:123', 'event:456']
```

Features

- Multiple granularity tracking
- Redis pipeline chaining
- Key scanner
- Easy to integrate with charting packages
- Can choose either integer or float counting
- Date bucketing with timezone support

Credits

Algorithm copied from tonyskn/node-redis-timeseries

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

Stable release

To install Redis Timeseries, run this command in your terminal:

```
$ pip install redis_timeseries
```

This is the preferred method to install Redis Timeseries, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

From sources

The sources for Redis Timeseries can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/ryananguiano/python-redis-timeseries
```

Or download the tarball:

```
$ curl  -OL https://github.com/ryananguiano/python-redis-timeseries/tarball/master
```

Once you have a copy of the source, you can install it with:

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

CHAPTER 3

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/ryananguiano/python-redis-timeseries/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

Write Documentation

Redis Timeseries could always use more documentation, whether as part of the official Redis Timeseries docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/ryananguiano/python-redis-timeseries/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *python-redis-timeseries* for local development.

1. Fork the *python-redis-timeseries* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/python-redis-timeseries.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv python-redis-timeseries
   $ cd python-redis-timeseries/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   ```
   $ flake8 redis_timeseries tests
   $ python setup.py test or py.test
   $ tox
   ```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/ryananguiano/python-redis-timeseries/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ py.test tests.test_redis_timeseries
```

CHAPTER 4

History

0.1.8 (2017-07-25)

- Fix bug in _round_time() method

0.1.7 (2017-07-25)

- Fix bug in _round_time() method

0.1.6 (2017-07-25)

- Add timezone so day buckets will start at midnight in the correct timezone

0.1.5 (2017-07-18)

- Update default granularities

0.1.4 (2017-07-12)

- Add float value capabilities
- Add increase() and decrease() methods
- Move get_hits() -> get_buckets() and get_total_hits() -> get_total()

0.1.3 (2017-03-30)

- Remove six package
- Clean up source file

0.1.2 (2017-03-30)

- Make Python 3 compatible
- Fix tox to make PyPy work

0.1.1 (2017-03-30)

- Minor project file updates

0.1.0 (2017-03-30)

- First release on PyPI.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search