django-redis-metrics Documentation

Release 1.0.0

Brad Montgomery

Jul 19, 2019

Contents

This app allows you do define various named metrics (such as 'New Users', 'Downloads', or 'Purchases'), record when they happen, and quickly and easily view that information.

Each metric can be assigned to an arbitrary category or given an optional expiration time. Granularity for metrics are saved daily, weekly, monthly, and yearly by default, but this libary gives you the ability to store event data down to the second, minute, and hour.

Here's a sneak peak at how it works:

```
>>> from redis_metrics.utils import metric
>>> metric("Downloads", category="User Metrics")
```

See the usage section for more examples.

CHAPTER 1

Inspiration

This app was inspired by Frank Wiles `django-app-metrics`. It offers a similar feature set but:

- It is *only* backed by Redis
- Does not require Celery
- Does not include any Timing

CHAPTER 2

License

This code is distributed under the terms of the MIT license. See the project's `LICENSE.txt` file for the full content of the license.

CHAPTER 3

Learn More

3.1 Installation

To get started quickly:

1. Run `pip install django-redis-metrics`.
2. Adding `redis_metrics` to your INSTALLED_APPS.
3. Include `url(r'^metrics/', include('redis_metrics.urls'))` in your URLConf.

3.1.1 Requirements

This app works with Python 3.6+ and is tested with Django 2.x. It also requires redis-py. For support for older versions of Django, see the 1.7.0 release.

If you'd like to run the tests, install the packages listed in `requirements/test.txt`, which includes coverage and mock.

3.1.2 Additional Installation

To install the current version, run `pip install django-redis-metrics`.

You can also install the development version with `pip install -e git://github.com/bradmontgomery/django-redis-metrics.git#egg=redis_metrics-dev`

To use the built-in views, add `redis_metrics` to your `INSTALLED_APPS`, and include the following in your Root URLconf:

```
from django.urls import re_path, include

re_path(r'^metrics/', include('redis_metrics.urls')),
```

Then, to view your metrics, visit the /metrics/ url, (i.e. run the development server and go to http://127.0.0.1:8000/metrics/)

3.1.3 Settings

Starting with version 1.0, this app contains a single setting, `REDIS_METRICS`, which includes a number of options, all wich have the following defaults:

```
REDIS_METRICS = {
    'CONNECTION_CLASS': None,
    'HOST': 'localhost',
    'PORT': 6379,
    'DB':  0,
    'PASSWORD': None,
    'SOCKET_TIMEOUT': None,
    'SOCKET_CONNECTION_POOL': None,
    'MIN_GRANULARITY': 'daily',
    'MAX_GRANULARITY': 'yearly',
    'MONDAY_FIRST_DAY_OF_WEEK': False,
    'USE_ISO_WEEK_NUMBER': False,
}
```

Formerly, each of these were separate settings with a `REDIS_METRICS_` prefix.

- **`CONNECTION_CLASS`: Optional name of a function which returns an instance of a Redis client.**
  - If you are using django-redis for caching, for example, then you can set this to `'django_redis.get_redis_connection'` to automatically use whatever django-redis is using to get its Redis client instances (django-redis supports pluggable back ends for Redis).
  - This can also be used to enable the use of Redis clients that support Redis Sentinel (e.g. use django-redis-sentinel along with django-redis), Redis Clustering or Hiredis, for example.
  - You can also create your own class/function that returns a valid Redis client.
  - **Note that if you specify this setting, then all other Redis-related settings will be ignored** (namely `HOST`, `PORT`, `DB`, `PASSWORD`, `SOCKET_TIMEOUT` and `SOCKET_CONNECTION_POOL`).
- `HOST`: Hostname of redis server that will contain your metrics data; defaults to 'localhost'
- `PORT`: Port of your redis server; defaults to '6379'
- `DB`: Your redis database number to use, defaults to 0
- `PASSWORD`: Your redis password if needed; defaults to None
- `SOCKET_TIMEOUT`: Your redis database socket timeout; defaults to None
- `SOCKET_CONNECTION_POOL`: Your redis database socket connection pool; defaults to None
- `MIN_GRANULARITY`: The minimum-time granularity for your metrics; default is 'daily'.
- `MAX_GRANULARITY`: The maximum-time granularity for your metrics; default is 'yearly'
- `MONDAY_FIRST_DAY_OF_WEEK`: Set to True if week should start on Monday; default is False
- `USE_ISO_WEEK_NUMBER`: Set to True to use ISO calendar weeks (see the isocalendar docs).

3.1.4 Upgrading versions prior to 0.8.x

If you used a version of this app prior to 0.8.0, it's likely that you'll run into this error:

```
WRONGTYPE Operation against a key holding the wrong kind of value
```

To fix this, run the `fix_redis_metrics_keys` command, which should migrate data for metrics, gauges, and categories into the format used by the current version of the app.

3.2 Usage

Use the `metric` shortcut to start recording metrics.

```
from redis_metrics import metric

# Increment the metric by one
metric('new-user-signup')

# Increment the metric by some other number
metric('new-user-signup', 4)
```

Metrics can also be categorized. To record a metric and add it to a category, specify a `category` keyword parameter

```
# Increment the metric, and add it to a category
metric('new-user-signup', category="User Metrics")
```

Metrics can also expire after a specified number of seconds

```
# The 'foo' metric will expire in 5 minutes
metric('foo', expire=300)
```

You can also *reset* a metric with the `set_metric` function. This will replace any existing values for the metric, rather than incrementing them. It's api is similar to `metric`'s.

```
from redis_metrics import set_metric

# Reset the Download count.
set_metric("downloads", 0)
```

3.2.1 Gauges

There are also `gauge`'s. A `gauge` is great for storing a *cumulative* value, and when you don't care about keeping a history for the metric. In other words, a gauge gives you a snapshot of some current value.

```
from redis_metrics import gauge

# Create a gauge
gauge('total-downloads', 0)

# Update the gauge
gauge('total-downloads', 9999)
```

3.2.2 The R class

There's also an `R` class which is a lightweight wrapper around `redis`. You can use it directly to set metrics or gauges and to retrieve data.

```
>>> from redis_metrics.models import R
>>> r = R()
>>> r.metric('new-user-signup')
>>> r.get_metric('new-user-signup')
{
    'second': 0,
    'minute': 0,
    'hour': 1,
    'day': '29',
    'month': '29',
    'week': '29',
    'year': '29'
}

# list the slugs you've used to create metrics
>>> r.metric_slugs()
set(['new-user-signup', 'user-logins'])

# Get metrics for multiple slugs
>>> r.get_metrics(['new-user-signup', 'user-logins'])
[
    {'new-user-signup': {
        'second': '0', 'minute': '0', 'hour': '1',
        'day': '7', 'month': '7', 'week': '7', 'year': '7'}},
    {'user-logins':
        'second': '0', 'minute': '0', 'hour': '1',
        'day': '7', 'month': '7', 'week': '7', 'year': '7'}},
]

# Delete a metric
>>> r.delete_metric("app-errors")
```

3.2.3 Templatetags

The included templatetags are useful for visualizing your stored metrics.

Load the templatetags in your template:: `{% load redis_metric_tags %}`

Viewing your data is possible with the built-in views, but these all make use of a number of templatetags to display metric data and history.

- `metrics_since(slugs, years, link_type="detail", granularity=None)` Renders a template with a menu to view a metric (or a list of metrics) for a given number of years. For example:

  ```
  {% metrics_since "downloads" 5 %}  {# downloads for the last 5 years #}
  ```

- `gauge(slug, maximum=9000, size=200, coerce='float')`: Includes a donut chart for the specified gauge. The maximum represents the largest possible value for the gague, while the size is the size of the chart in pixels. The coerce parameter tells the template tag how to coerce numeric data. By default values are converted to floats, but you can include `coerce='int'` to force values to be listed as integers.:

  ```
  {% gauge "tasks-completed" 10 size=300 coerce='int' %}
  ```

- `metric_list` generates a list of all metrics.
- `metric_detail(slug, with_data_table=False)` displays a metric's current details. This tag will also generate a table of raw data if the `with_data_table` option is True.

- `metric_history(slug, granularity="daily", since=None, to=None, with_data_table=False)` displays a given metric's history. The `granularity` option defines the granularity displayed, `since` is a string or datetime object that specifies the date and/or time from which we start displaying data, the `to` argument indicates to date or time to which we display data, and `with_data_table` controls wether or not raw data is displayed in a table. Examples:

```
{# dainly signups since Jan 1, 2015 #}
{% metric_history "signups" "daily" "2015-01-01" %}

{# daily signups between Jan 1, 2015 & Jan, 1 2016 #}
{% metric_history "signups" "daily" "2015-01-01" "2016-01-01" %}

{# monthly signups for a given year #}
{% metric_history "signups" "monthly" this_year %}


{# monthly signups for a given year, including data  #}
{% metric_history "signups" "monthly" this_year with_data_table=True %}
```

- `aggregate_detail(slug_list, with_data_table=False)` is much like `metric_detail`, but displayes more than one metric on the chart. The `slug_list` parameter should be a list of metric slugs that you want to display.
- `aggregate_history(slug_list, granularity="daily", since=None, with_data_table=False)` is similarly like `metric_history`, but for multiple metrics on once chart. but displayes more than one metric on the chart. The `slug_list` parameter should be a list of metric slugs that you want to display.

3.3 Contributing

Ideas and pull requests are welcome! Please feel free to submit bug reports or pull requests on the github repo.

If you do submit a pull request, please consider running the tests and (if applicable) adding test coverage for your changes. This project uses travis to automatically run the test suite for all pull requests. There's also an `AUTHORS.rst` file containing all of the people who have contributed code. Please add yourself.

Thank You in advance!

CHAPTER 4

Indices and tables

- genindex
- modindex
- search