django-statsd Documentation

Release 0.3

Andy McKay

Jun 14, 2017

Contents

CHAPTER 1

Django Statsd

Integration between statsd and django. It allows you to use different clients, sends timings as middleware and integrates with django debug toolbar.

Credits:

- jbalogh and jsocol for statsd and commonware, which I just ripped parts out of and put in here.
- robhudson for django-debug-toolbar

Changes

0.3.15:

- push from travis to pypi to keep files clean
- allow less statsd in the middleware
- fix to a specific statsd version

0.3.14:

- pypy testing support
- log model changes
- log celery events
- log db queries
- show lower/mean/upper values in debugbar, thanks jonathanslenders!

0.3.12:

- Event better Django 1.6 support for the patches, with tests.

0.3.11:

- Django 1.6 support

0.3.9:

- statsd 2.0 support
- improved Django debug toolbar support

0.3.8.5:

- don't count some 404 as 500 and fix deprecation warnings

0.3.8.4:

- gauge support

0.3.8.3:

- some bug fixes

0.3.8.1:

- add in a tasty pie middleware

0.3.8:

- add in a nose plugin

0.3.7:

- add in metlog client

0.3.6:

- add in log handler

0.3.5:

- fix tests to work standalone
- add in waterfall view of timings

0.3.3:

- fix setup.py to include loggers etc

0.3.2:

- update to work with latest Django Debug Toolbar

0.3:

- added in logging handler for logging error counts to stats

Installation

From pypi:

```
pip install django-statsd-mozilla
```

Because there is already a django-statsd on pypi.

Requirement, https://github.com/jsocol/pystatsd or:

```
pip install statsd
```

Because there is already a pystatsd on pypi. This will be automatically added when you install django-statsd-mozilla.

First off, pick your client, one of:

- django_statsd.clients.null

  This one does nothing, good for development. No point in wasting those UDP packets.

- django_statsd.clients.toolbar

  Use for the django debug toolbar, stores all the statsd pings on the request so they can be used in the toolbar.

- django_statsd.clients.normal

  Use this for production, it just passes through to the real actual pystatsd.

- django_statsd.clients.log

  Just writes the values to a log file using Python's logging module.

- django_statsd.clients.moz_metlog

  Use this to route messages through _metlog: http://github.com/mozilla-services/metlog-py. Note that using metlog will require you to bind the metlog instance to bind the metlog client instance as settings.METLOG.

- django_statsd.clients.nose

  Route messages through to the nose plugin. This also works with the toolbar client, so you don't need to change them on -dev.

Usage

To send timings from your code, use just like pystatsd, but change your imports to read:

```
from django_statsd.clients import statsd
```

For example:

```
from django_statsd.clients import statsd
statsd.incr('response.200')
```

Django statsd will choose the client as specified in your config and send the data to it. You can change your client by specifying it in the config, the default is:

```
STATSD_CLIENT = 'django_statsd.clients.normal'
```

To send timings or counts with every request, add in some middleware:

```
MIDDLEWARE_CLASSES = (
        'django_statsd.middleware.GraphiteRequestTimingMiddleware',
        'django_statsd.middleware.GraphiteMiddleware',
        ) + MIDDLEWARE_CLASSES
```

If you are using tastypie, you might enjoy:

```
'django_statsd.middleware.TastyPieRequestTimingMiddleware'
```

To get timings for your database or your cache, put in some monkeypatches:

```
STATSD_PATCHES = [
        'django_statsd.patches.db',
        'django_statsd.patches.cache',
]
```

You can change the host that stats are sent to with the *STATSD_HOST* setting:

```
STATSD_HOST = 'localhost'
```

Similarly, you can use the *STATSD_PORT'setting to customize the port number (which defaults to '8125*):

```
STATSD_PORT = 8125
```

Toolbar integration

Make sure *django_statsd* is installed:

```
INSTALLED_APPS = (
        ..
        'django_statsd',
)
```

This will show you the statsd timings in the toolbar:

```
MIDDLEWARE_CLASSES = (
        'debug_toolbar.middleware.DebugToolbarMiddleware',
        ) + MIDDLEWARE_CLASSES
```

Note: this must go before the GraphiteMiddleware so that we've got the timing data in before we show the toolbar panel.

Add in the panel:

```
DEBUG_TOOLBAR_PANELS = (
     ...
     'django_statsd.panel.StatsdPanel',
)
```

Set the client:

```
STATSD_CLIENT = 'django_statsd.clients.toolbar'
```

Finally if you have production data coming into a graphite server, you can show data from that server. If you have one, link it up:

Here's the configuration we use on AMO. Because triggers and counts go to different spots, you can configure them differently:

```
TOOLBAR_STATSD = {
        'graphite': 'https://graphite-phx.mozilla.org/render/',
        'roots': {
                'timers': ['stats.timers.addons-dev', 'stats.timers.addons'],
                'counts': ['stats.addons-dev', 'stats.addons']
        }
}
```

The key is added on to the root. So if you've got a key of *view.GET* this would look that up on the graphite server with the key:

```
stats.addons.view.GET
```

Django Model save and delete integration

You can log all create, update and delete events of django models. Add to your Django settings:

```
STATSD_MODEL_SIGNALS = True
```

Celery signals integration

You can log all the `task_sent`, `task_prerun`, `task_postrun` and `task_failure` signals of celery along with the duration of succesful tasks.

To enable this, add the following to your Django settings:

```
STATSD_CELERY_SIGNALS = True
```

Front end timing integration

New browsers come with an API to provide timing information, see:

http://w3c-test.org/webperf/specs/NavigationTiming/

To record this in statsd you need a JavaScript lib on the front end to send data to the server. You then use the server to record the information. This library provides a view to hook that up for different libraries.

First, make sure you can record the timings in your Django site urls. This could be done by pointing straight to the view or including the URL for example:

```
from django_statsd.urls import urlpatterns as statsd_patterns

urlpatterns = [
    url(r'^services/timing/', include(statsd_patterns)),
]
```

In this case the URL to the record view will be */services/timing/record*.

Second, hook up the client. There is a un-sophisticated client called *stick* included in the static directory. This requires no configuration on your part, just make sure that the file *django_statsd/static/stick.js* is in your sites JS.

Then call it in the following manner:

```
stick.send('/services/timing/record');
```

We also include support for *boomerang*, a sophisticated client from Yahoo:

http://yahoo.github.com/boomerang

To hook this up, first add in boomerang to your site, make sure you use the web timing enabled version, as discussed here:

http://yahoo.github.com/boomerang/doc/howtos/howto-9.html

When the script is added to your site, add the following JS:

```
BOOMR.init({
        beacon_url: '/services/timing/record'
}).addVar('client', 'boomerang');
```

Once you've installed either boomerang or stick, you'll see the following keys sent:

```
window.performance.timing.domComplete 5309|ms
window.performance.timing.domInteractive 3819|ms
window.performance.timing.domLoading 1780|ms
window.performance.navigation.redirectCount 0|c
window.performance.navigation.type.reload 1|c
```

There's a couple of options with this you can set in settings:

STATSD_RECORD_KEYS (optional)

A list of the keys you want to record, there's quite a few in the timing api and you likely don't want to record them all. Here's the default:

```
STATSD_RECORD_KEYS = [
        'window.performance.timing.domComplete',
        'window.performance.timing.domInteractive',
        'window.performance.timing.domLoading',
        'window.performance.navigation.redirectCount',
        'window.performance.navigation.type',
]
```

Override this to get different ones.

STATSD_RECORD_GUARD (optional)

There's only limited ways to stop people posting junk to your URLs. By defining a this a function you can do some work to allow requests to your needs. If the function returns None, the request is allowed through. If you don't want to allow the request, return any valid Django HTTP response. For example to deny everyone not in INTERNAL_IPS:

```
from django.http import HttpResponseForbidden

def internal_only(request):
    if request.META['REMOTE_ADDR'] not in INTERNAL_IPS:
        return HttpResponseForbidden()

STATSD_RECORD_GUARD = internal_only
```

STATSD_VIEW_TIMER_DETAILS (optional)

The middleware sends timing pings for the almost the same thing three times when accessing a view: *module.name.method*, *module.method* and *method* by default. Setting this to *False* just does the former.

Logging errors

If you want to log a count of the errors in your application to statsd, you can do this by adding in the handler. For example in your logging configuration:

```
'handlers': {
    'test_statsd_handler': {
        'class': 'django_statsd.loggers.errors.StatsdHandler',
    },
}
```

CHAPTER 2

Testing

You need to install tox to run the tests. You can run the full test matrix with:

```
tox
```

or choose a specific environment - let's say Python 3.4 and Django 1.11 - with:

```
tox -e py34-django111
```

You can list all the available environments with:

```
tox -l
```

CHAPTER 3

Nose

There is also a nose plugin. If you use nose, then run tests, you'll get output in your tests. To use run tests with the following:

```
--with-statsd
```

- streeter
- crankycoder
- glogiotatidis
- tominsam
- youngbob
- jsatt
- youngbob
- jsocol
- janfabry
- tomchristie
- diox
- frewsxcv
- fud
- ftobia
- jawnb
- fgallina
- jonathanslenders
- streeter

See:

https://github.com/andymckay/django-statsd/pulls?direction=desc&page=1&sort=created&state=closed

CHAPTER 4

Indices and tables

- genindex
- modindex
- search