django-watchman Documentation

Release 0.15.0

Michael Warkentin

Mar 05, 2018

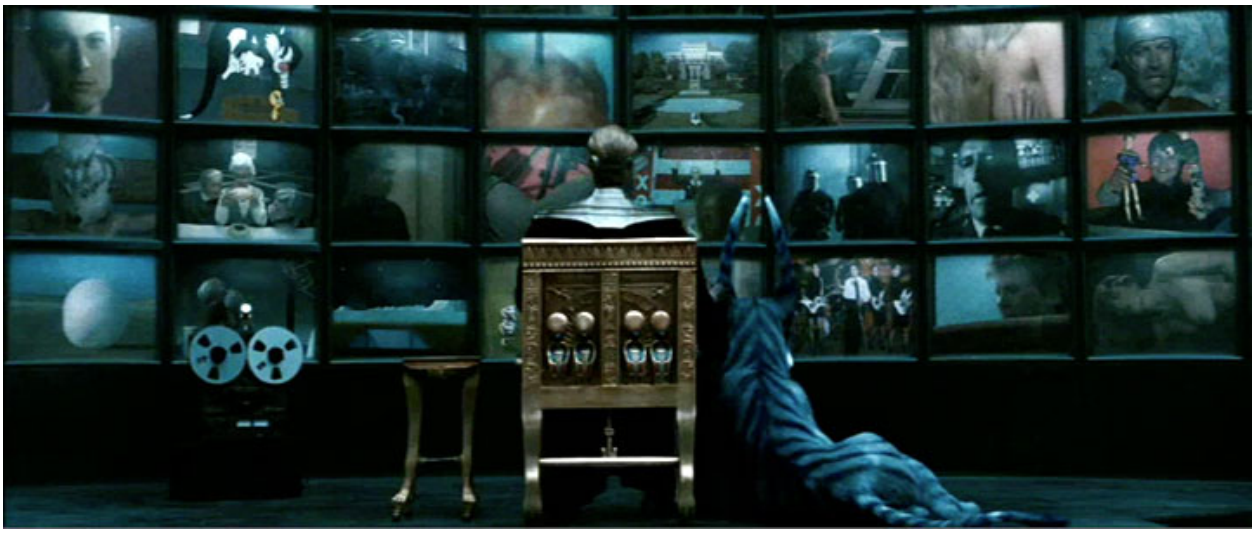
Contents

Contents:

CHAPTER 1

django-watchman

django-watchman exposes a status endpoint for your backing services like databases, caches, etc.



1.1 Documentation

The full documentation is at http://django-watchman.rtfd.org.

1.2 Testimonials

> We're in love with django-watchman. External monitoring is a vital part of our service offering. Using django-watchman we can introspect the infrastructure of an application via a secure URL. It's very well written and easy to extend. We've recommended it to many of our clients already.

— Hany Fahim, CEO, VM Farms.

1.3 Quickstart

1. Install `django-watchman`:

```
pip install django-watchman
```

2. Add `watchman` to your `INSTALLED_APPS` setting like this:

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

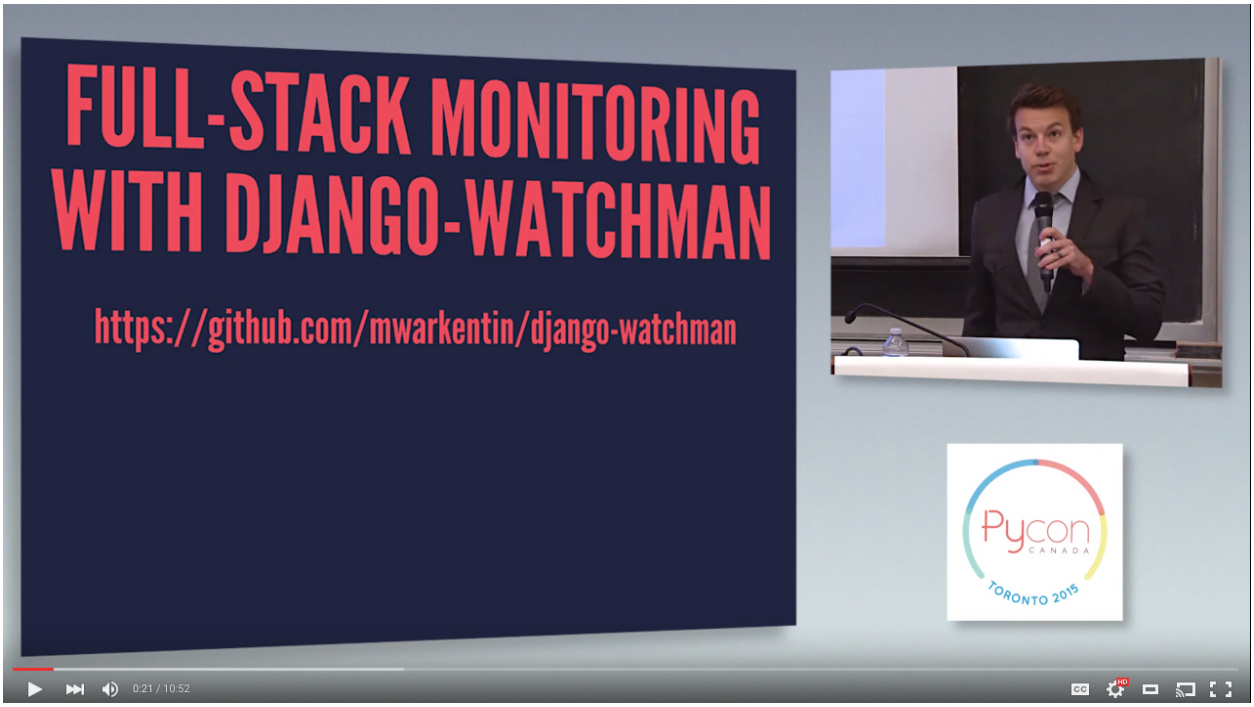
3. Include the watchman URLconf in your project `urls.py` like this:

```
url(r'^watchman/', include('watchman.urls')),
```

4. Start the development server and visit `http://127.0.0.1:8000/watchman/` to get a JSON response of your backing service statuses:

```
{
    "databases": [
        {
            "default": {
                "ok": true
            }
        }
    ],
    "caches": [
        {
            "default": {
                "ok": true
            }
        }
    ],
    "storage": {"ok": true}
}
```

1.4 Pycon Canada Presentation (10 minutes)



1.5 Features

1.5.1 Human-friendly dashboard

Visit `http://127.0.0.1:8000/watchman/dashboard/` to get a human-friendly HTML representation of all of your watchman checks.

1.5.2 Token based authentication

If you want to protect the status endpoint, you can use the `WATCHMAN_TOKENS` setting. This is a comma-separated list of tokens. When this setting is added, you must pass one of the tokens in as the `watchman-token` **GET** parameter:

```
GET http://127.0.0.1:8000/watchman/?watchman-token=:token
```

Or by setting the `Authorization: WATCHMAN-TOKEN` header on the request:

```
curl -X GET -H "Authorization: WATCHMAN-TOKEN Token=\":token\"" http://127.0.0.1:8000/
↪watchman/
```

If you want to change the token name, you can set the `WATCHMAN_TOKEN_NAME`. The value of this setting will be the **GET** parameter that you must pass in:

```
WATCHMAN_TOKEN_NAME = 'custom-token-name'

GET http://127.0.0.1:8000/watchman/?custom-token-name=:token
```

DEPRECATION WARNING: `WATCHMAN_TOKEN` was replaced by the `WATCHMAN_TOKENS` setting to support multiple authentication tokens in django-watchman `0.11`. It will continue to work until it's removed in django-watchman `1.0`.

1.5.3 Custom authentication/authorization

If you want to protect the status endpoint with a customized authentication/authorization decorator, you can add `WATCHMAN_AUTH_DECORATOR` to your settings. This needs to be a dotted-path to a decorator, and defaults to `watchman.decorators.token_required`:

```
WATCHMAN_AUTH_DECORATOR = 'django.contrib.admin.views.decorators.staff_member_required
↪'
```

Note that the `token_required` decorator does not protect a view unless `WATCHMAN_TOKENS` is set in settings.

1.5.4 Custom checks

django-watchman allows you to customize the checks which are run by modifying the `WATCHMAN_CHECKS` setting. In `settings.py`:

```
WATCHMAN_CHECKS = (
    'module.path.to.callable',
    'another.module.path.to.callable',
)
```

Checks take no arguments, and must return a `dict` whose keys are applied to the JSON response. Use the `watchman.decorators.check` decorator to capture exceptions:

```
from watchman.decorators import check

@check
def my_check():
    return {'x': 1}
```

In the absence of any checks, a 404 is thrown, which is then handled by the `json_view` decorator.

1.5.5 Run a subset of available checks

A subset of checks may be run, by passing `?check=module.path.to.callable&check=...` in the request URL. Only the callables given in the querystring which are also in `WATCHMAN_CHECKS` should be run, eg:

```
curl -XGET http://127.0.0.1:8080/watchman/?check=watchman.checks.caches
```

1.5.6 Skip specific checks

You can skip any number of checks, by passing `?skip=module.path.to.callable&skip=...` in the request URL. Only the checks in `WATCHMAN_CHECKS` which are not in the querystring should be run, eg:

```
curl -XGET http://127.0.0.1:8080/watchman/?skip=watchman.checks.email
```

1.5.7 Check a subset of databases or caches

If your application has a large number of databases or caches configured, watchman may open too many connections as it checks each database or cache.

You can set the `WATCHMAN_DATABASES` or `WATCHMAN_CACHES` settings in order to override the default set of databases and caches to be monitored.

1.5.8 Ping

If you want to simply check that your application is running and able to handle requests, you can call ping:

GET http://127.0.0.1:8000/watchman/ping/

It will return the text `pong` with a 200 status code. Calling this doesn't run any of the checks.

1.5.9 Bare status view

If you would like a "bare" status view (one that doesn't report any details, just `HTTP 200` if checks pass, and `HTTP 500` if any checks fail), you can use the `bare_status` view by putting the following into `urls.py`:

```
import watchman.views
# ...
url(r'^status/?$', watchman.views.bare_status),
```

1.5.10 Django management command

You can also run your checks without starting the webserver and making requests. This can be useful for testing your configuration before enabling a server, checking configuration on worker servers, etc. Run the management command like so:

```
python manage.py watchman
```

By default, successful checks will not print any output. If all checks pass successfully, the exit code will be `0`. If a check fails, the exit code will be `1`, and the error message including stack trace will be printed to `stderr`.

If you'd like to see output for successful checks as well, set verbosity to `2` or higher:

```
python manage.py watchman -v 2
{"storage": {"ok": true}}
{"caches": [{"default": {"ok": true}}]}
{"databases": [{"default": {"ok": true}}]}
```

If you'd like to run a subset of checks, use `-c` and a comma-separated list of python module paths:

```
python manage.py watchman -c watchman.checks.caches,watchman.checks.databases -v 2
{"caches": [{"default": {"ok": true}}]}
{"databases": [{"default": {"ok": true}}]}
```

If you'd like to skip certain checks, use `-s` and a comma-separated list of python module paths:

```
python manage.py watchman -s watchman.checks.caches,watchman.checks.databases -v 2
{"storage": {"ok": true}}
```

Use `-h` to see a full list of options:

```
python manage.py watchman -h
```

1.5.11 X-Watchman-Version response header

Watchman can return the version of watchman which is running to help you keep track of whether or not your sites are using an up-to-date version. This is disabled by default to prevent any unintended information leakage for websites without authentication. To enable, update the `EXPOSE_WATCHMAN_VERSION` setting:

```
EXPOSE_WATCHMAN_VERSION = True
```

1.5.12 Custom response code

By default, watchman will return a `500` HTTP response code, even if there's a failing check. You can specify a different response code for failing checks using the `WATCHMAN_ERROR_CODE` setting:

```
WATCHMAN_ERROR_CODE = 200
```

1.5.13 Logging

watchman includes log messages using a logger called `watchman`. You can configure this by configuring the `LOGGING` section of your Django settings file.

Here is a simple example that would log to the console:

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'console': {
            'class': 'logging.StreamHandler',
        },
    },
    'loggers': {
        'watchman': {
            'handlers': ['console'],
            'level': 'DEBUG',
        },
    },
}
```

More information is available in the Django documentation.

1.5.14 APM (i.e. New Relic)

If you're using APM and watchman is being often hit for health checks (such as an ELB on AWS), you will find some stats based on averages will be affected (average transaction time, apdex, etc):

You can disable APM instrumentation for watchman by using the `WATCHMAN_DISABLE_APM` setting:

```
WATCHMAN_DISABLE_APM = True
```

This currently supports the following agents:

- New Relic

Please open an issue if there's another APM you use which is being affected.

1.6 Available checks

1.6.1 caches

For each cache in `django.conf.settings.CACHES`:

- Set a test cache item
- Get test item
- Delete test item

1.6.2 databases

For each database in `django.conf.settings.DATABASES`:

- Verify connection by calling `connections[database].introspection.table_names()`

1.6.3 email

Send a test email to `to@example.com` using `django.core.mail.send_mail`.

If you're using a 3rd party mail provider, this check could end up costing you money, depending how aggressive you are with your monitoring. For this reason, this check is **not enabled** by default.

For reference, if you were using Mandrill, and hitting your watchman endpoint once per minute, this would cost you ~$5.60/month.

Custom Settings

- `WATCHMAN_EMAIL_SENDER` (default: `watchman@example.com`): Specify an email to be the sender of the test email
- `WATCHMAN_EMAIL_RECIPIENTS` (default: `[to@example.com]`): Specify a list of email addresses to send the test email
- `WATCHMAN_EMAIL_HEADERS` (default: `{}`): Specify a dict of custom headers to be added to the test email

1.6.4 storage

Using `django.core.files.storage.default_storage`:

- Write a test file
- Check the test file's size
- Read the test file's contents
- Delete the test file

1.6.5 Default checks

By default, django-watchman will run checks against your databases (`watchman.checks.databases`), caches (`watchman.checks.caches`), and storage (`watchman.checks.storage`).

1.6.6 Paid checks

Currently there is only one "paid" check - `watchman.checks.email`. You can enable it by setting the `WATCHMAN_ENABLE_PAID_CHECKS` to `True`, or by overriding the `WATCHMAN_CHECKS` setting.

1.7 Trying it out with Docker

A sample project is available along with a Dockerfile to make it easy to try out django-watchman.

1.7.1 Requirements

- *Docker <https://www.docker.com/get-docker>*

1.7.2 Instructions

1. Build and run the Docker image with the current local code: `make run`
2. Visit watchman json endpoint in your browser: http://127.0.0.1:8000/watchman/
3. Visit watchman dashboard in your browser: http://127.0.0.1:8000/watchman/dashboard/
4. Visit watchman ping in your browser: http://127.0.0.1:8000/watchman/ping/
5. Visit watchman bare status in your browser: http://127.0.0.1:8000/watchman/bare/

CHAPTER 2

Installation

At the command line:

```
$ easy_install django-watchman
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv django-watchman
$ pip install django-watchman
```

CHAPTER 3

Usage

To use django-watchman in a project:

```
import django-watchman
```

CHAPTER 4

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:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/mwarkentin/django-watchman/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.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

4.1.4 Write Documentation

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

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/mwarkentin/django-watchman/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 :)

4.2 Get Started!

Ready to contribute? Here's how to set up *django-watchman* for local development.

1. Fork the *django-watchman* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/django-watchman.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 django-watchman
$ cd django-watchman/
$ python setup.py develop
```

4. Create a branch for local development:

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

a. 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 watchman tests
$ python setup.py test
$ tox

a. 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.

4.3 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, and 3.3, and for PyPy. Check https://travis-ci.org/mwarkentin/django-watchman/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_watchman
```

CHAPTER 5

Credits

5.1 Author / Maintainer

- Michael Warkentin <mwarkentin@gmail.com> - https://github.com/mwarkentin - @mwarkentin

5.2 Contributors

- Keryn Knight - https://github.com/kezabelle
- blag - https://github.com/blag
- kilbasar - https://github.com/kilbasar
- Joseph Kahn - https://github.com/JBKahn
- Ben Webber - https://github.com/benwebber
- Michael Fladischer - https://github.com/fladi
- Justin Sacbibit - https://github.com/justinsacbibit
- Ulrich Petri - https://github.com/ulope
- Tim Tisdall - https://github.com/tisdall
- Eduardo Cardoso - https://github.com/eduardocardoso
- Daniel Widerin - https://github.com/saily
- Ryan Wilson-Perkin - https://github.com/ryanwilsonperkin
- David Hoffman - https://github.com/dhoffman34
- James M. Allen - https://github.com/jamesmallen
- Ryan Verner - https://github.com/xfxf

CHAPTER 6

History

6.1 0.15.0 (2018-02-27)

- [#114] Add "bare" status view (@jamesmallen)
- [#115] Adds `WATCHMAN_DISABLE_APM` option (@xfxf)
- [#63] Disable watchman version output by default, add `EXPOSE_WATCHMAN_VERSION` setting (@mwarkentin)

6.2 0.14.0 (2018-01-09)

- [#110] Replace vagrant + ansible with Dockerfile (@ryanwilsonperkin)
- [#111] Configure Django logging for checks (@dhoffman34)
- [#112] Add simple HTTP ping endpoint (@dhoffman34)

6.3 0.13.1 (2017-05-27)

- [#101] Write `bytes` to dummy file on storage check to fix an issue in Python 3 (thanks @saily!)

6.4 0.13.0 (2017-05-23)

- [#105] Add `WATCHMAN_CACHES` and `WATCHMAN_DATABASES` settings to override the Django defaults
  - When using watchman with a large number of databases, the default checks can cause an excess of connections to the database / cache
  - New settings allow you to check only a subset of databases / caches

  - Watchman will still default to checking all databases / caches, so no changes necessary for most apps

6.5 0.12.0 (2017-02-22)

- [#100] Add `WATCHMAN_EMAIL_SENDER` setting to customize email check "from" address

6.6 0.11.1 (2017-02-14)

- [#99] Fix verbose output in management command on Django 1.8+

6.7 0.11.0 (2016-08-02)

- Update tests to run on Django 1.7 - 1.10
- [#87] Fix 500 errors with ATOMIC_REQUESTS enabled
  - Disables atomic transactions on the watchman views to prevent generic 500 errors
- [#88] Restructure dashboard and switch icon libraries
  - Make check types singular on dashboard
  - Switch to FontAwesome instead of Glyphicon to track Bootstrap updates
  - Improve traceback display width
- [#92] Support multiple auth tokens
  - Fixes [#86]
  - Deprecates `settings.WATCHMAN_TOKEN` and adds `settings.WATCHMAN_TOKENS`

6.8 0.10.1 (2016-05-03)

- [#81] Fix header-based authentication for tokens w/ dashes (-)
  - Regex was overly specific for header values (*w*)
  - Added TODO to follow up with a full regex for valid characters according to the spec

6.9 0.10.0 (2016-05-02)

- [#75] Enable header-based authentication
  - Set a header instead of passing the token via GET param: `"Authorization: WATCHMAN-TOKEN Token=\":token\""`
  - Improves security by keeping tokens out of logs
- [#79] Enable customization of email check
  - Add `WATCHMAN_EMAIL_RECIPIENTS` setting - pass a list of recipients the email should be sent to
  - Add `WATCHMAN_EMAIL_HEADERS` setting - pass a dict of custom headers to be set on the email

6.10 0.9.0 (2015-12-16)

- [#51] Update TravisCI Python / Django versions
- [#52] Fix deprecated `url_patterns`
- [#53] Change default error response code to `500`
- [#56] Add `@check` decorator and refactor existing checks to use it (thanks @benwebber!)
- [#57] Sort `caches` / `databases` in response for more consistent responses
- [#59] Add `.editorconfig` for improved consistency in contributions
- [#61] Add `Vagrantfile` and docs for how to run and develop on Vagrant instance
- [#65] Include assets in source tarball for Debian packaging (thanks @fladi)
- [#71] Unpin *django-jsonview* in setup.py
- [#72] Fix stacktrace on dashboard modal and increase width for better readability

6.11 0.8.0 (2015-10-03)

- [#46] Allow custom response codes with the `WATCHMAN_ERROR_CODE` setting

6.12 0.7.1 (2015-08-14)

- Update headers in `HISTORY.rst` to attempt to fix localshop parsing issues

6.13 0.7.0 (2015-08-14)

- [#40] Bump `django-jsonview` for improved Django 1.8 compatibility
  - Also brought travis Django test versions in line with currently supported Django versions (1.4.x, 1.7.x, 1.8.x)

6.14 0.6.0 (2015-07-02)

- [#30] Allow users to specify a custom authentication/authorization decorator
  - Override the `@auth` decorator by setting `WATCHMAN_AUTH_DECORATOR` to a dot-separated path to your own decorator
  - eg. `WATCHMAN_AUTH_DECORATOR = 'django.contrib.admin.views.decorators.staff_member_required'`
  - Token-based authentication remains the default
- [#31], [#34] Add a human-friendly status dashboard
  - Available at `<watchman url>/dashboard/`
  - `?check` & `?skip` GET params work on the dashboard as well

- [#35] Add `X-Watchman-Version` header to responses

6.15 0.5.0 (2015-01-25)

- Add `watchman` management command
  - Exit code of `0` if all checks pass, `1` otherwise
  - Print json stacktrace to `stderr` if check fails
  - Handles `--verbosity` option to print all status checks
  - `-c`, `--checks`, `-s`, `--skips` options take comma-separated list of python paths to run / skip
- Improve identifiability of emails sent from a django-watchman endpoint
  - From: watchman@example.com
  - Subject: django-watchman email check
  - Body: This is an automated test of the email system.
  - Add `X-DJANGO-WATCHMAN: True` custom header
- Add new default check: `storage` check
  - Checks that files can be both written and read with the current Django storage engine
  - Add `WATCHMAN_ENABLE_PAID_CHECKS` setting to enable all paid checks without modifying `WATCHMAN_CHECKS`
- Remove `email_status` from default checks
- Refactor `utils.get_checks` to allow reuse in management command
  - `get_checks` now performs the optional check inclusion / skipping
  - `status` refactored to pull `check_list` / `skip_list` from GET params and pass them to `get_checks`
- Namespace cache keys
- Update documentation

6.16 0.4.0 (2014-09-08)

- Add the ability to skip certain checks by passing one or more `skip=path.to.callable` GET params when hitting the watchman URL

6.17 0.3.0 (2014-09-05)

- New check - email (`watchman.checks.email_status`)! django-watchman will now check that your email settings are working too!
- Fix a few small issues in the readme
- Rearrange some of the code in checks.py

6.18 0.2.2 (2014-09-05)

- Fix and run tests on Python 2.7 and 3.4
- Bump django-jsonview dependency to latest
- Update tox envlist and travis config to test 2.7 / 3.4

6.19 0.2.1 (2014-09-04)

- Initialize django during tests to prevent app loading issue for Django >= 1.7
- Suppress `MIDDLEWARE_CLASSES` warning for Django >= 1.7
- Reorganize test imports
- Fix `make test`, `make coverage`, `make release` commands
- Add htmlcov/ directory to .gitignore
- Test django 1.4, 1.6, 1.7

6.20 0.2.0 (2014-09-04)

- Custom checks can now be written and run using the `WATCHMAN_CHECKS` setting
- A subset of the available checks can be run by passing the `check` GET param when hitting the watchman url

6.21 0.1.2 (2014-02-21)

- Move package requirements out of requirements.txt and into setup.py

6.22 0.1.1 (2014-02-09)

- Remove `django>=1.5.5` version specification
- Remove `wheel` requirement

6.23 0.1.0 (2014-02-08)

- First release on PyPI.