kardboard Documentation

Release 1.17

Chris Heisel

March 13, 2015

Contents:

Kanban concepts

kardboard is meant to represent physical cards on a Kanban board. To understand the application it's useful to have some background on Kanban concepts.

Card A card represents a unit of work, it's the central object around which almost everything else revolves. Ultimatley it represents some piece of business value.

Ticket For kardboard's purposes a ticket is an item contained in a seperate system, that has additional data about a particular card. For example, a Card in kardboard may know when a card was backlogged, started and completed but its Ticket would know who it's assigned to currently. A Card and a Ticket are linked together via the Card's **key** value.

Backlog A card that is still in a planning, or Todo state. Backlog is one of three core states a card can be in.

In progress / work in progress Cards that moved from the backlog to being actively worked on by someone. Typically "In progress" covers many sub-states in a teams workflow.

Done Cards for whom the value has been delivered, by whatever criteria the team uses. For software development teams it could mean the software was deployed, or signed off on and is ready for release, etc.

Lead time A measurement, in number of business days, between the date a card enters the Backlog and the date the card is Done. This a measure of the time it takes from "we decided to do a thing" to "the thing we decided to do is done." Think about rides at Disney World. Lead time is a measurement of how long it takes from you waiting in line till the time you step off the ride.

Cycle time A measurement, in number of business days, between the date a card enters In progress to the date the card is Done. This is a measure of time it takes from "we started doing a thing" to "the thing is done.". Think about rides at Disney World. Cycle time is a measurement of how long it takes from when you get on the ride to when you get off.

Throughput The number of cards that can be completed in a given time frame. The lower the cycle time, the higher the throughput. In Disney World terms, it's riders-per-hour.

States The user-defined steps cards go through on their way from the Backlog to Done. For some teams this may be a single step "Doing", for others it can be a very complex workflow. See *CARD_STATES*

Team If you're managing multiple development teams, you can use the Team concept in kardboard to have what are essentially "mini-boards" for each team while still seeing a "meta-board" that shows you cards across all states from all teams. See *CARD_TEAMS*

CHAPTER 2

Installing Kardboard

2.1 Settings & Configuration

Much of kardboard can be controlled via settings. The easiest way control these is add an environment variable, **KARDBOARD_SETTINGS**, that is a path to a .cfg file containing these settings.

```
export KARDBOARD_SETTINGS=/opt/www/kardboardve/etc/kardboard-prod.conf
```

The config file should be in a format compatible with Flask's file-based configuration standard.

2.1.1 Flask settings

Kardboard is built atop Flask and as such all of its built in options can be adjusted in your configuration file.

2.1.2 Kardboard settings

All the necessary settings to get a basic kardboard instance up and running are contained in kardboard/default_settings.py

CACHE_TYPE

Default: `'simple'` Specifies which type of caching object to use. This is an import string that will be imported and instantiated. It is assumed that the import object is a function that will return a cache object that adheres to the werkzeug cache API.

For werkzeug.contrib.cache objects, you do not need to specify the entire import string, just one of the following names.

Built-in cache types:

- **null**: NullCache
- **simple**: SimpleCache
- **memcached**: MemcachedCache
- **gaememcached**: GAEMemcachedCache
- **filesystem**: FileSystemCache

CACHE_ARGS

Default: `[]` (Empty list)

Optional list to unpack and pass during the cache class instantiation.

CACHE_OPTIONS

Default: `{}` (Empty dictionary)

Optional dictionary to pass during the cache class instantiation.

CACHE_DEFAULT_TIMEOUT

Default: `3600`

The default timeout that is used if no timeout is specified. Unit of time is seconds.

CACHE_THRESHOLD

Default: (No default)

The maximum number of items the cache will store before it starts deleting some. Used only for SimpleCache and FileSystemCache

CACHE_KEY_PREFIX

Default: (No default)

A prefix that is added before all keys. This makes it possible to use the same memcached server for different apps. Used only for MemcachedCache and GAEMemcachedCache.

CACHE_MEMCACHED_SERVERS

Default: (No default)

A list or a tuple of server addresses. Used only for MemcachedCache

CACHE_DIR

Default: (No default)

Directory to store cache. Used only for FileSystemCache.

CARD_CATEGORIES

Default: `[ 'Bug', 'Feature', 'Improvement',]`

The list of categories users should be able to assign cards to. Example:

```
CARD_CATEGORIES = [ 'CMS', 'iPhone', 'Android', ]
CARD_CATEGORIES = [ 'Project X', 'Project Y', 'Project Z']
```

CARD_STATES

Default: `[ 'Todo', 'Doing', 'Done', ]`

The list of states, or columns, that a card could be in. **The last state should represent whatever Done means to your team.**

Tip: When a user sets a Done date for a card, it's automatically set to the last state in your CARD_STATES setting.

CARD_TEAMS

Default: `[ ('Team 1', ), ('Team 2', 10), ]`

The list of teams, and optionally their work-in-progress limits working on cards. This allows you to have "mini-boards" for each team/person while still seeing a "meta-board" that shows you cards across all states from all teams.

CYCLE_TIME_GOAL

Default: No default

A Python two-element tuple containing the lower and upper bounds of a cycle time goal for the board. Used to highlight cards in the range and beyond the range.:

```
CYCYLE_TIME_GOAL = (10, 15)
CYCYLE_TIME_GOAL = (15, 15)
```

CELERYD_LOG_LEVEL

Default: `'INFO'`

See Celery configuration documentation for details

BROKER_TRANSPORT

Default: `'redis'`

See Celery configuration documentation for details

CELERY_RESULT_BACKEND

Default: `'redis'`

See Celery configuration documentation for details

CELERY_IMPORTS

Default: `('kardboard.tasks', )`

See Celery configuration documentation for details

CELERYBEAT_SCHEDULE

Default:

```
{
    # How often should we look for old tickets and queue them for updates
    'load-update-queue': {
        'task': 'tasks.queue_updates',
        'schedule': crontab(minute="*/3"),
    },
    # How often should we update all the Person
    # objects to make sure they reflect reality, due to deleted cards
    # or people being removed from a card
    'update_person': {
        'task': 'tasks.normalize_people',
        'schedule': crontab(minute="*/30"),
    },
    # How often (probably nighly) should we update daily records for the past
    # 365 days
    'calc-daily-records-year': {
        'task': 'tasks.update_daily_records',
        'schedule': crontab(minute=1, hour=0),
        'args': (365, ),
    },
    # How often should we update daily records for the past
    # 7 days
    'calc-daily-records-week': {
        'task': 'tasks.update_daily_records',
        'schedule': crontab(minute="*/5"),
        'args': (14, ),
    }
}
```

If you're using a *TICKET_HELPER* then you probably don't want to adjust this setting. The *crontab(minute="*/3")* determines how often kardboard should check for out of date cards. See *TICKET_UPDATE_THRESHOLD* for more.

See Celery configuration documentation for details

GOOGLE_SITE_VERIFICATION

Default: (No default)

If set, it will output an appropriate <meta> tag so you may claim your site on Google Webmaster Tools.

```
GOOGLE_SITE_VERIFICATION = 'someverylongstringgoeshere'
```

GOOGLE_ANALYTICS

Default: (No default)

If set, it will output an appropriate <script> tag for Google Analytics.

```
GOOGLE_ANALYTICS = 'UA-11111111-2'
```

JIRA_CREDENTIALS

Default: (No default)

A two item tuple consisting of a username and password that has at least read-only access to any projects and tickets you'll be entering into kardboard.

```
JIRA_CREDENTIALS = ('jbluth', 'theresalwaysmoneyinthebananastand')
```

JIRA_WSDL

Default: (No default)

If you set *TICKET_HELPER* to use the built-in JIRAHelper then you'll want to set this to your JIRA installation's SOAP end point.

```
JIRA_WSDL = 'https://jira.yourdomain.com/rpc/soap/jirasoapservice-v2?wsdl'
```

LOG_LEVEL

Default: (No default)

The level of log events that should be output to *LOG_FILE*.

Possible settings are:

- `'debug'`
- `'info'`
- `'warning'`
- `'critical'`
- `'error'`

LOG_FILE

Default: (No default)

The file that log events should be written too.

```
LOG_FILE = '/var/logs/kardboard-app.log'
```

Note: The LOG_FILE file will be automatically rotated every ~100k and up to 3 previous ~100k chunks will be kept.

MONGODB_DB

Default: `'kardboard'`

The name of the database you want to store your data in.

MONGODB_PORT

Default: `27017`

The port MongoDB is running on.

SECRET_KEY

Default: `'yougonnawannachangethis'`

A secret key for this particular kardboard instance. Used to provide a seed in secret-key hashing algorithms. Set this to a random string – the longer, the better.

As the default implies, you're going to want to change this.

TICKET_AUTH

Default: `False`

If True, then add/edit/delete views become protected by authentication tied to your TICKET_HELPER instance.

So for example if you had the following:: TICKET_HELPER = 'kardboard.tickethelpers.JIRAHelper'
  TICKET_AUTH = True

Then users would be required to login with their JIRA credentials.

TICKET_HELPER

Default: `'kardboard.tickethelpers.NullHelper'`

A Python class that will fetch additional information from a ticketing system (JIRA, Redmine, Pivotal Tracker, e.g.) about a card.

The only provider shipped with kardboard is `'kardboard.tickethelpers.JIRAHelper'`.

TICKET_UPDATE_THRESHOLD

Default: `60*5` (seconds)

The minimum length of time **in seconds** before a individual card has its data updated from its ticketing system of record.

Every 90 seconds (unless changed in *CELERYBEAT_SCHEDULE*), kardboard will scan for cards older than *TICKET_UPDATE_THRESHOLD* and fetch data on them.

CHAPTER 3

Developing kardboard

3.1 Using Vagrant

The way to develop on kardboard is via Vagrant , powered by the included Puppet files.

3.1.1 Default setup

```
# Get the source, using your own fork most likely
git clone git@github.com:cmheisel/kardboard.git

cd kardboard
vagrant up
# ...wait...
vagrant ssh
touch kardboardve/src/kardboard/kardboard-local.cfg
source kardboardve/bin/activate
cd kardboardve/src/kardboard
py.test kardboard
```

You now have a fully functional kardboard application running under production like environment.

- The application is served by gunicorn and nginx and is available at http://localhost:8080/ from your host machine. It runs at http://localhost:80/ on the guest.
- Memcache is running and used by the application
- Redis is running and used as the queue for your running celery process
- Both the application and celery processes are adminsitered through supervisord
- You can restart both the application and the celery processes, say to pick up your changes, by doing the following:

```
vagrant ssh
sudo supervisorctl restart all
```

- Logs for the application, gunicorn, nginx and celery are in /home/vagrant/logs/

3.1.2 Development server

Testing against a production like environment is all well and good for you, like vegetables. But during development you'll likely want something tastier, like cookies, or a auto-reloading web server.

```
vagrant ssh
export KARDBOARD_SETTINGS=/vagrant/kardboard-local.cfg
cd /home/vagrant/kardboardve/
source bin/activate
python src/kardboard/kardboard/runserver.py
```

The application is now available at http://localhost:5000/ from your host machine.

CHAPTER 4

API

4.1 Kard

4.2 Ticket Helpers

- *modindex*