kozmic Documentation

Release 0.0.1

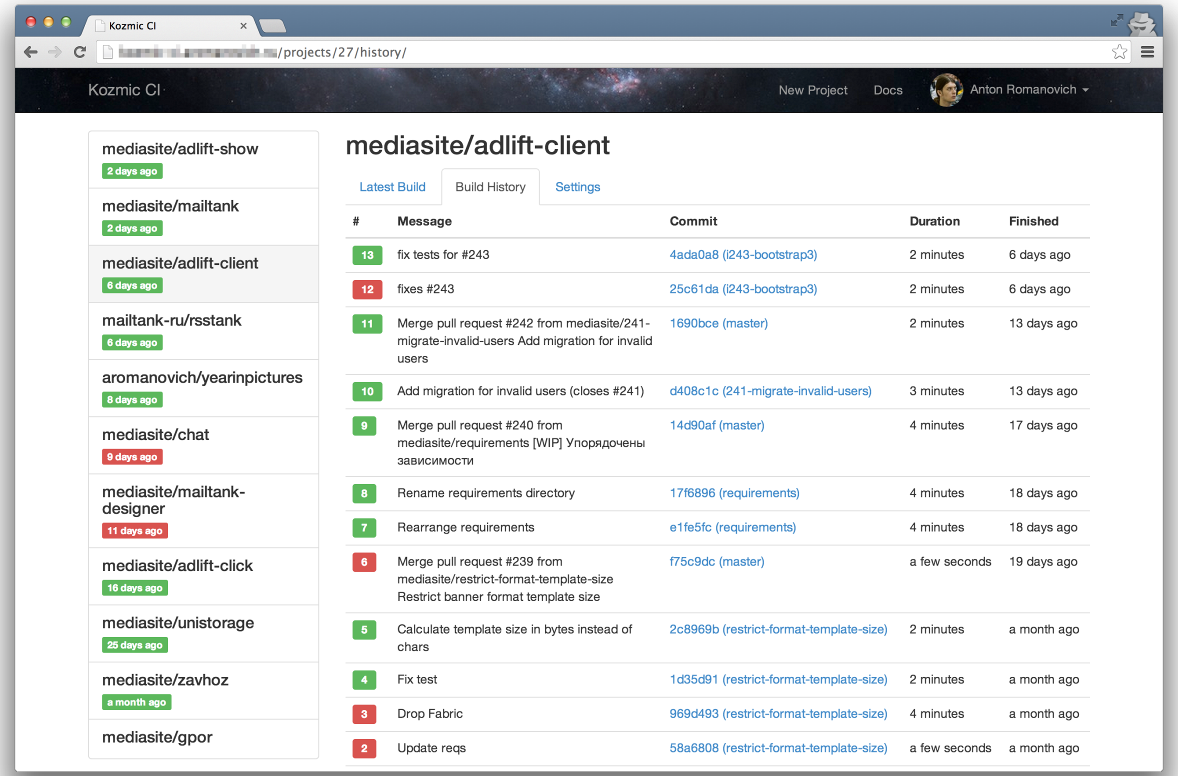
Anton Romanovich

June 14, 2016

Contents

Kozmic CI is a self-hosted continuous integration service. It is written in Python, integrated with GitHub and powered by Docker.

Thanks to Docker, it provides immutable build environments and powerful build dependencies caching mechanism.



mediasite/adlift-show
2 days ago

mediasite/mailtank
2 days ago

mediasite/adlift-client
6 days ago

mailtank-ru/rsstank
6 days ago

aromanovich/yearinpictures
8 days ago

mediasite/chat
9 days ago

mediasite/mailtank-designer
11 days ago

mediasite/adlift-click
16 days ago

mediasite/unistorage
25 days ago

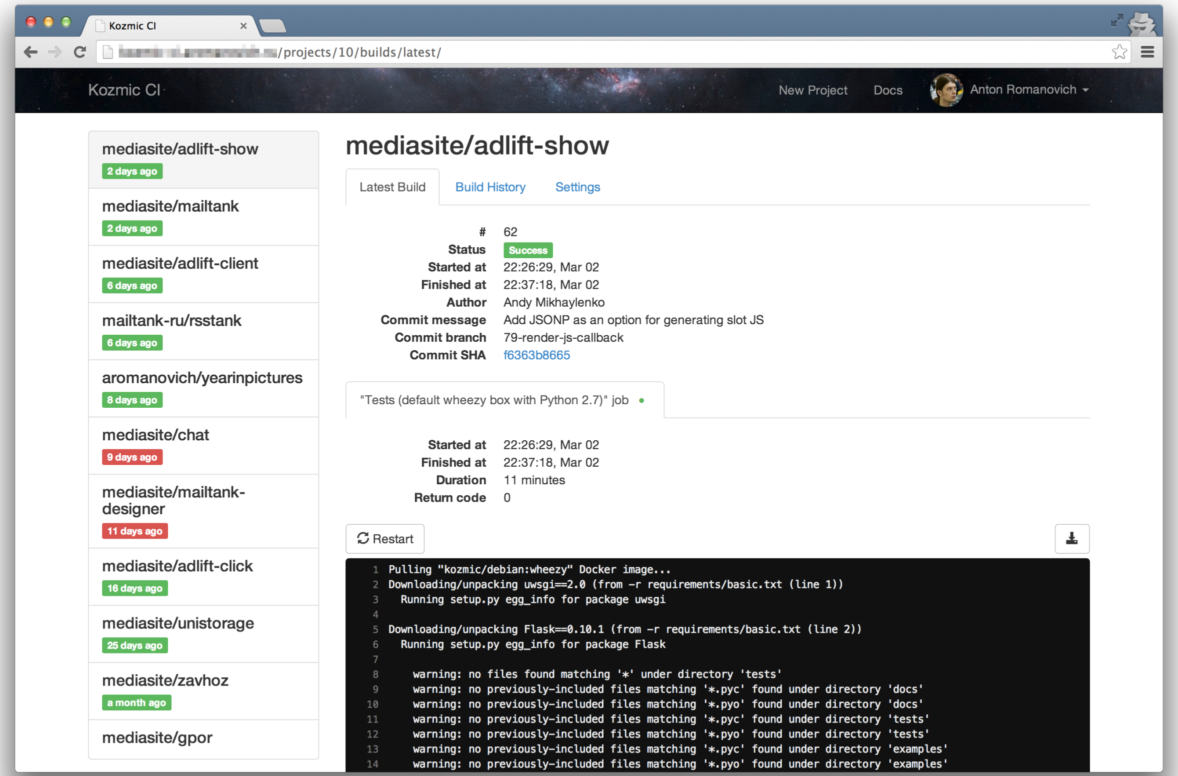
mediasite/zavhoz
a month ago

mediasite/gpor

mediasite/adlift-client

Latest Build | Build History | Settings

#	Message	Commit	Duration	Finished
13	fix tests for #243	4ada0a8 (i243-bootstrap3)	2 minutes	6 days ago
12	fixes #243	25c61da (i243-bootstrap3)	2 minutes	6 days ago
11	Merge pull request #242 from mediasite/241-migrate-invalid-users Add migration for invalid users	1690bce (master)	2 minutes	13 days ago
10	Add migration for invalid users (closes #241)	d408c1c (241-migrate-invalid-users)	3 minutes	13 days ago
9	Merge pull request #240 from mediasite/requirements [WIP] Упорядочены зависимости	14d90af (master)	4 minutes	17 days ago
8	Rename requirements directory	17f6896 (requirements)	4 minutes	18 days ago
7	Rearrange requirements	e1fe5fc (requirements)	4 minutes	18 days ago
6	Merge pull request #239 from mediasite/restrict-format-template-size Restrict banner format template size	f75c9dc (master)	a few seconds	19 days ago
5	Calculate template size in bytes instead of chars	2c8969b (restrict-format-template-size)	2 minutes	a month ago
4	Fix test	1d35d91 (restrict-format-template-size)	2 minutes	a month ago
3	Drop Fabric	969d493 (restrict-format-template-size)	4 minutes	a month ago
2	Update reqs	58a6808 (restrict-format-template-size)	a few seconds	a month ago



mediasite/adlift-show
2 days ago
mediasite/mailtank
2 days ago
mediasite/adlift-client
6 days ago
mailtank-ru/rsstank
6 days ago
aromanovich/yearinpictures
8 days ago
mediasite/chat
9 days ago
mediasite/mailtank-designer
11 days ago
mediasite/adlift-click
16 days ago
mediasite/unistorage
25 days ago
mediasite/zavhoz
a month ago
mediasite/gpor

mediasite/adlift-show

Latest Build | Build History | Settings

#	62
Status	Success
Started at	22:26:29, Mar 02
Finished at	22:37:18, Mar 02
Author	Andy Mikhaylenko
Commit message	Add JSONP as an option for generating slot JS
Commit branch	79-render-js-callback
Commit SHA	f6363b8665

"Tests (default wheezy box with Python 2.7)" job ●

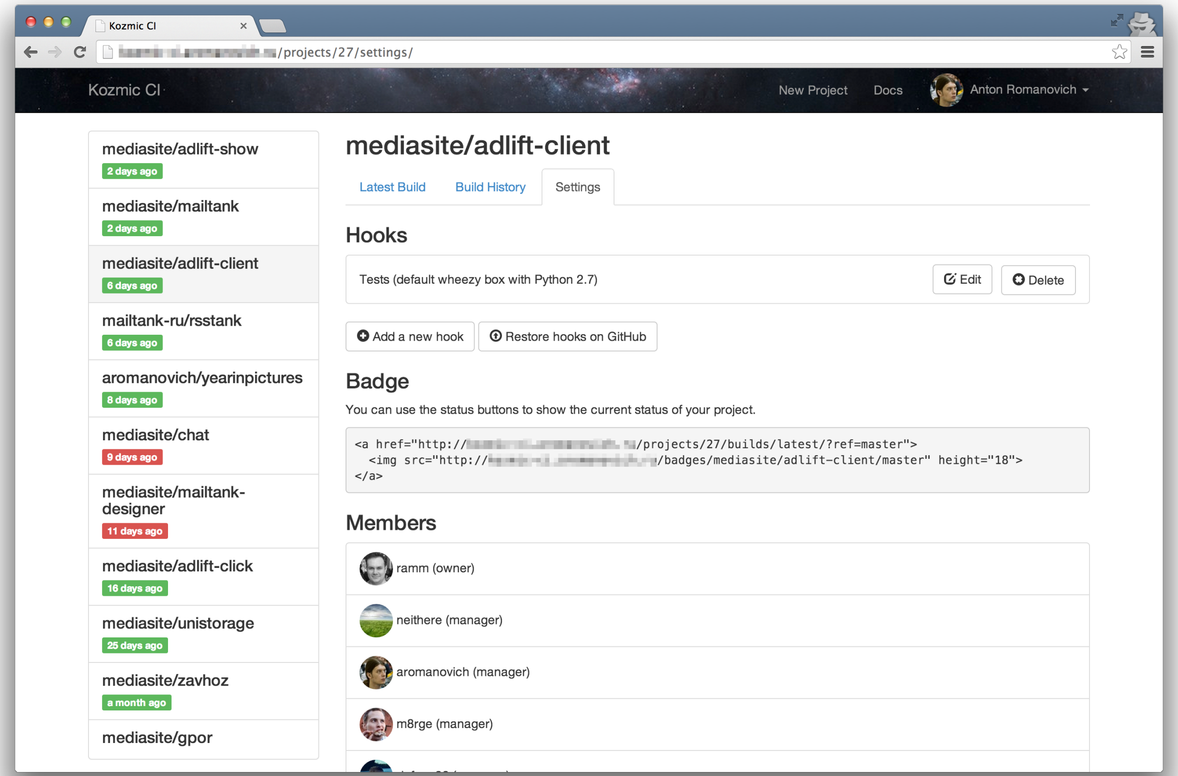
Started at	22:26:29, Mar 02
Finished at	22:37:18, Mar 02
Duration	11 minutes
Return code	0

Restart

```
Pulling "kozmic/debian:wheezy" Docker image...
Downloading/unpacking uwsgi==2.0 (from -r requirements/basic.txt (line 1))
  Running setup.py egg_info for package uwsgi

Downloading/unpacking Flask==0.10.1 (from -r requirements/basic.txt (line 2))
  Running setup.py egg_info for package Flask

    warning: no files found matching '*' under directory 'tests'
    warning: no previously-included files matching '*.pyc' found under directory 'docs'
    warning: no previously-included files matching '*.pyo' found under directory 'docs'
    warning: no previously-included files matching '*.pyc' found under directory 'tests'
    warning: no previously-included files matching '*.pyo' found under directory 'tests'
    warning: no previously-included files matching '*.pyc' found under directory 'examples'
    warning: no previously-included files matching '*.pyo' found under directory 'examples'
```



mediasite/adlift-show
2 days ago

mediasite/mailtank
2 days ago

mediasite/adlift-client
6 days ago

mailtank-ru/rsstank
6 days ago

aromanovich/yearinpictures
8 days ago

mediasite/chat
9 days ago

mediasite/mailtank-designer
11 days ago

mediasite/adlift-click
16 days ago

mediasite/unistorage
25 days ago

mediasite/zavhoz
a month ago

mediasite/gpor

mediasite/adlift-client

Latest Build | Build History | Settings

Hooks

Tests (default wheezy box with Python 2.7) — Edit — Delete

Add a new hook — Restore hooks on GitHub

Badge

You can use the status buttons to show the current status of your project.

```
<a href="http://[redacted]/projects/27/builds/latest/?ref=master">
  <img src="http://[redacted]/badges/mediasite/adlift-client/master" height="18">
</a>
```

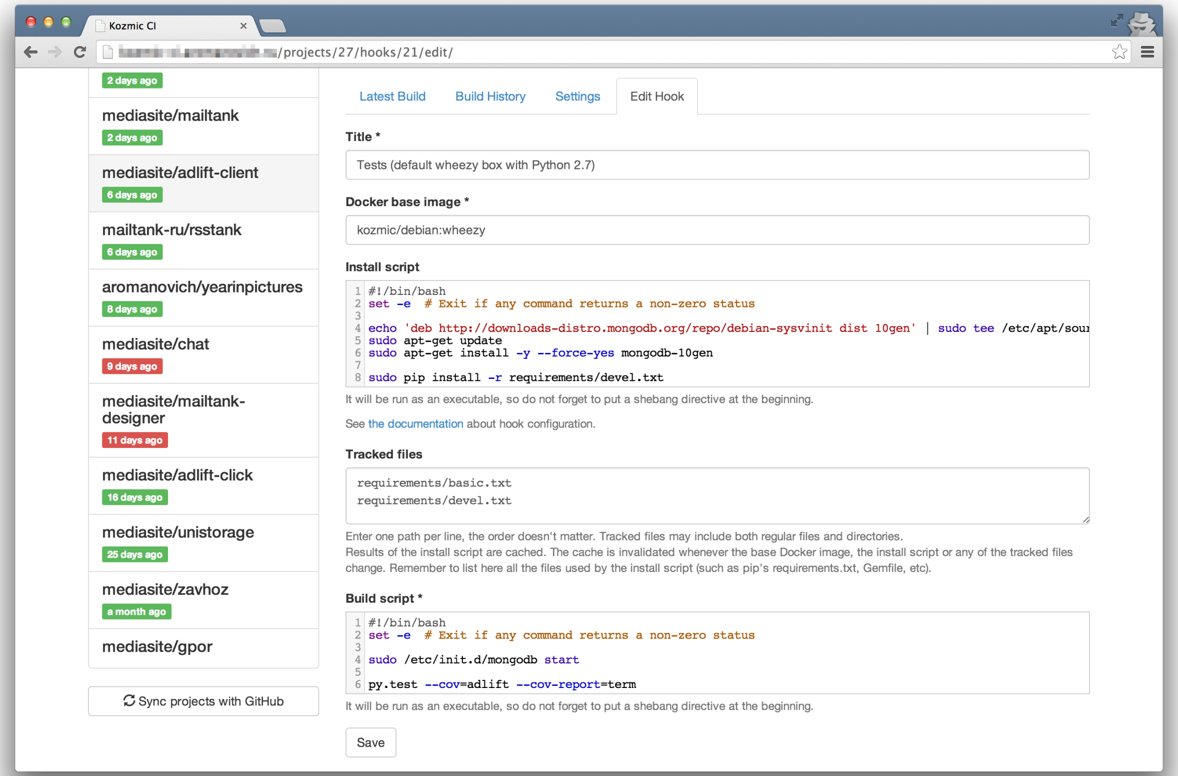
Members

ramm (owner)

neithere (manager)

aromanovich (manager)

m8rge (manager)



Kozmic CI

[redacted]/projects/27/hooks/21/edit/

2 days ago

mediasite/mailtank
2 days ago

mediasite/adlift-client
6 days ago

mailtank-ru/rsstank
6 days ago

aromanovich/yearinpictures
8 days ago

mediasite/chat
9 days ago

mediasite/mailtank-designer
11 days ago

mediasite/adlift-click
16 days ago

mediasite/unistorage
25 days ago

mediasite/zavhoz
a month ago

mediasite/gpor

Sync projects with GitHub

Latest Build | Build History | Settings | Edit Hook

Title *

Tests (default wheezy box with Python 2.7)

Docker base image *

kozmic/debian:wheezy

Install script

```
#!/bin/bash
set -e  # Exit if any command returns a non-zero status

echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' | sudo tee /etc/apt/sour
sudo apt-get update
sudo apt-get install -y --force-yes mongodb-10gen

sudo pip install -r requirements/devel.txt
```

It will be run as an executable, so do not forget to put a shebang directive at the beginning.

See the documentation about hook configuration.

Tracked files

```
requirements/basic.txt
requirements/devel.txt
```

Enter one path per line, the order doesn't matter. Tracked files may include both regular files and directories.
Results of the install script are cached. The cache is invalidated whenever the base Docker image, the install script or any of the tracked files change. Remember to list here all the files used by the install script (such as pip's requirements.txt, Gemfile, etc).

Build script *

```
#!/bin/bash
set -e  # Exit if any command returns a non-zero status

sudo /etc/init.d/mongodb start

py.test --cov=adlift --cov-report=term
```

It will be run as an executable, so do not forget to put a shebang directive at the beginning.

Save

CHAPTER 1

System Overview

1.1 What is Kozmic CI?

Kozmic CI is a Docker powered continuous integration platform integrated with GitHub.

It is written in Python using Flask and Celery. It uses Docker for a job isolation and dependencies caching, MySQL as a data storage, Redis as a pub-sub implementation and uWSGI as a websockets framework.

1.2 Why Kozmic CI?

There are plenty of continuous integration tools out there and they all have pros and cons.

Some of them are very powerful but rather complex to use, like Jenkins CI or TeamCity. Sometimes something simpler would be enough. Something like Travis CI seems a good way to go until you don't constantly find yourself littering the commit history trying to debug the over-complicated .travis.yml, or want to use a custom VM image for your builds.

Kozmic CI is intended to be somewhere in between: to be easy to set up on your own server, configure and use, but flexible enough to be capable of performing any kind of job.

And hey, it is powered by Docker! Docker is cool! :)

1.3 Kozmic CI Philosophy

Kozmic CI is made with simplicity in mind.

It doesn't do much by itself. It delegates job isolation and dependencies caching to Docker. It uses GitHub as an authentication and authorization provider. It doesn't maintain any VM images with pre-installed languages and databases. It doesn't even introduce a build configuration format.

You are free to use your favorite scripting language to describe a build. You'll probably have to learn some Bash while writing build scripts, but in return you'll be given control, predictability and ease of debugging.

The lack of "official" images with pre-installed stuff is a deliberate choice. You have to set up the environment yourself — it encourages you to keep your testing environment close to the production one and pin your requirements. And then you'll not one day be surprised by a broken build when the official VM image is upgraded.

1.4 Basics

Kozmic CI is tightly integrated with GitHub.

There are **users** and **projects**. Kozmic users correspond to GitHub users, Kozmic projects correspond GitHub repositories.

A user can have either one of the following roles.

- An owner, can view, configure and delete the project
- A manager, can view and configure the project
- A member, can only view the project

Projects' memberships are determined by GitHub permissions.

- The owner is the user who created the project
- Managers are those users who can push and pull from the GitHub repository
- Members are those users who can only pull from the GitHub repository

A project can have one or more **hooks** which map one-to-one to GitHub webhooks. A Kozmic hook defines a **job** to be performed when the corresponging GitHub hook is triggered.

A Kozmic **build** is basically a set of the jobs triggered by the same GitHub commit. If all the jobs have succeeded, the build is considered successful. If any of the jobs has failed, the build is considered failed. A build status is reported to GitHub as a Commit Status.

1.5 More About Hooks and Jobs

As it has been mentioned above, hooks describe jobs.

To configure a hook, you must specify a Docker **base image** and a **build script**. A build script is just an executable. It must start with a shebang sequence (i.e., `#!/bin/bash`) and everything that follows is completely up to you. You can use your favorite scripting language: bash, Python, Perl, basically anything that present in the base image.

In short, what Kozmic CI will do is to pull that Docker image from Central Registry and run the build script in it. The job is considered successful when the build script exits with zero return code and failed otherwise.

Also you can specify an **install script** and it's **tracked files**.

The install script is an executable, much like the build script. Tracked files are a list of paths in the repository.

The install script runs before the build script. The result of a running the install script, a Docker container, is promoted to a Docker image and cached. During the next job, if neither the install script or it's tracked filed have changed, the install script will be skipped and the cached image will be reused for running the build script.

That provides a really powerful tool for caching dependencies.

1.6 Base Images

Kozmic CI runs builds in isolated Docker containers that offer a clean environment for every build.

These containers are created using base images. A base image is a Docker image that meets a few requirements:

1. It must have the following packages installed:

- `bash`

   - `sudo`
   - `git`
   - `openssh-client`

2. It must have a user named `kozmic` with sudo rights without password check

At this point Kozmic CI supports base images that are only hosted on a Central Registry provided by the Docker project.

To tell Kozmic CI use particular base image for running a job, you must specify it's repository name in the hook settings. Repository names look like `<username>/<repo_name>`, i.e. `kozmic/ubuntu-base`. You can also specify a tag from that repository, i.e. `kozmic/ubuntu-base:12.04`.

The specified base image will be pulled from the registry before running the first job.

Kozmic CI provides a number of "official" base images: https://index.docker.io/u/kozmic/. They are all built using Trusted Build service and their Dockerfiles are hosted on GitHub. If some of the base images is missing something, or you built a base image for your own needs and think that it may be useful for others – please feel free to submit a pull request or open an issue.

CHAPTER 2

Installation and Set Up

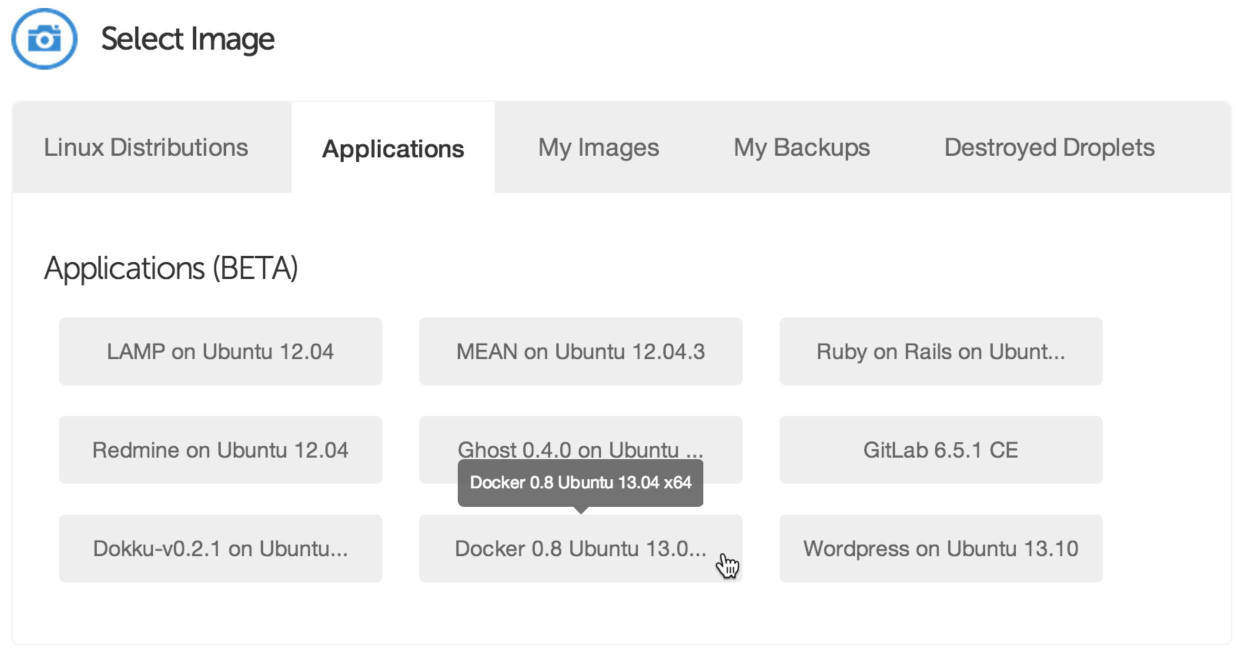
2.1 The Fast Way

Kozmic CI offers a Docker-based single-node distribution.

It has some limitations, but it's the fastest and easiest way to get started.

2.1.1 Step 1: Install Docker

If you use Digital Ocean, you can just create a droplet from an image with pre-installed Docker:



If you use Ubuntu 13.04 or later, installing Docker is just as simple as that:

```
apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 36A1D7869245C8950F966E92D8576A8BA88D21E9
echo deb http://get.docker.io/ubuntu docker main > /etc/apt/sources.list.d/docker.list
apt-get update
apt-get install -f lxc-docker
```

If you use another OS, take a look at Docker installation instructions.

2.1.2 Step 2: Register a new application on GitHub

Go to https://github.com/settings/applications/new and create a new application.

Set the homepage URL to `http://my-server-ip-or-addr` and the authorization callback URL to `http://my-server-ip-or-addr/_auth/auth-callback`.

2.1.3 Step 3: Start Kozmic CI

Create a directory for Kozmic CI logs:

```
mkdir -p $HOME/kozmic-ci/log
```

Create a data-only container that will be used to persist the Kozmic CI data:

```
docker run -v /var/lib/docker -v /var/lib/mysql --name kozmic-data ubuntu:12.04 true
```

Run Kozmic CI:

```
JOB=$(docker run -e=SECRET_KEY=xxxxx \
                 -e=GITHUB_CLIENT_ID=xxxxxxxxxxxxxxxxxxxx \
                 -e=GITHUB_CLIENT_SECRET=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx \
                 -e=SERVER_NAME=xxx.xxx.xxx.xxx \
                 -p=80:80 \
                 -p=8080:8080 \
                 --volumes-from kozmic-data \
                 -v=$HOME/kozmic-ci/log:/var/log \
                 --privileged -d aromanovich/kozmic:0.0.1)
docker logs $JOB
```

A few comments:

- `SECRET_KEY` must be set to a unique, unpredictable value. It *must* be kept the same if you are restarting or updating Kozmic CI container.
- `GITHUB_CLIENT_ID` and `GITHUB_CLIENT_SECRET` must contain the OAuth client id and secret of your GitHub application.
- `SERVER_NAME` must contain an IP address or domain name of the machine. It must be accessible from the outside Internet.
- `-p=80:80 -p=8080:8080` binds the container ports to the host system.
- `-v=$HOME/kozmic-ci/log:/var/log` mounts the directory from the host into the container which make is possible to see what's going on inside.
- `--privileged` key is required for running Docker within Docker.

After starting the container, take a look at the `logs` directory content and make sure that it doesn't say any errors. That's it!

2.2 The Usual Way

The usual way is to not use Docker-based distribution, but manually deploy each of the three components:

- A web application that implements UI and exposes webhooks (`kozmic`)
- A uWSGI-application that sends a job log into a websocket (`tailer`)

- A Celery-worker that runs jobs

These components require Python 2.7, MySQL, Redis and Docker.

A Kozmic CI's Dockerfile is pretty much self-documenting about how to deploy them.

It uses Supervisor for running all the components (see the last three sections of supervisor.conf) and uWSGI as an application server for `kozmic` and `tailer` (see kozmic-uwsgi.ini and tailer-uwsgi.ini).

You will also have to use `manage.py` to run the database migrations:

```
KOZMIC_CONFIG=kozmic.config_local.Config ./manage.py db upgrade
```

If you're planning to use Kozmic CI status images in GitHub README files, they must be served through HTTPS to prevent GitHub from caching them (see *KOZMIC_USE_HTTPS_FOR_BADGES* setting).

`tailer` **must** be run using uWSGI that is listed in its requirements (`./requirements/tailer.txt`).

CHAPTER 3

Reference

It's important to understand how jobs are performed in order to efficiently use Kozmic CI features such as dependencies caching.

As it has been mentioned earlier, Kozmic CI uses Docker for a job isolation and dependencies caching.

A job is defined by a hook. A hook consists of:

- Docker base image
- Build script
- Install script (optional)
- Tracked files (optional)

3.1 Job Workflow

Here's what Kozmic CI does when GitHub triggers the hook.

- The Docker base image is pulled from the Central registry.
- If the install script is specified and it hasn't been run before or if the base image, the install script itself or some of tracked files have been changed, the install script is run and the resulting container is promoted to an image and cached.

  Otherwise this step is skipped.
- The build script is run in a Docker container created either from a cached image (if the install script is specified) or Docker base image.

If either the install script or the build script exits with a return code different from zero, the job considered failed.

3.2 How Scripts Are Run

Install scripts are processed the same way as build scripts. The only difference is that a result of an install script, a container, is cached.

1. A container is created from that image. It's `/kozmic` directory is a volume and mounted to the host machine.
2. The script to be run is placed in that directory, along with some auxiliary files: a helper for running the script, a file to which the script output will be written, deploy key, etc.
3. If the project's repository is private, `ssh-agent` is started and the private deploy key is added to it.

4. The repository is cloned to `/kozmic/src` and the required commit is checked out.
5. Finally, the script is run in the `/kozmic/src` directory from the `kozmic` user. `/kozmic` directory and it's content owned by `kozmic` user.

Note: Changes that the install script makes to the `/kozmic` directory will not be cached.

3.3 Examples

3.3.1 MySQL and Python

Suppose the project is written in Python and uses MySQL. Here's an example of a hook configuration.

Docker base image: `kozmic/ubuntu:12.04`.

Install script:

```
#!/bin/bash
set -e  # Exit if any command returns a non-zero status

sudo su <<EOF
pip install -r ./requirements/basic.txt
pip install -r ./requirements/dev.txt
EOF
```

Tracked files:

```
requirements/basic.txt
requirements/dev.txt
```

Build script:

```
#!/bin/bash
set -e

sudo su <<EOF
/usr/bin/mysqld_safe &
sleep 3  # Give it time to start
mysql -e 'create database rsstank_test character set utf8 collate utf8_general_ci;'
EOF

cp ./rsstank/config_local.py-kozmic ./rsstank/config_local.py
./test.sh
```

We run `pip` from root because it sets up packages system-wide.

MySQL is already set up in the `kozmic/ubuntu:12.04` image. It has to be started manually before the tests, because Docker doesn't use Ubuntu's init system.

3.3.2 MongoDB and Python

Here's another example for a project that uses MongoDB.

Docker base image: `kozmic/debian:wheezy`.

Install script:

```
#!/bin/bash
set -e  # Exit if any command returns a non-zero status

echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' | \
    sudo tee /etc/apt/sources.list.d/mongodb.list
sudo apt-get update
sudo apt-get install -y --force-yes mongodb-10gen

sudo pip install -r requirements/devel.txt
```

Tracked files:

```
requirements/basic.txt
requirements/dev.txt
```

Build script:

```
#!/bin/bash
set -e  # Exit if any command returns a non-zero status

sudo /etc/init.d/mongodb start

py.test --cov=adlift --cov-report=term
```

CHAPTER 4

Configuration

An environment variable `KOZMIC_CONFIG` tells the application (`kozmic.create_app()` and `tailer`) which config to use. For example, to run a development server you can use the following command: `KOZMIC_CONFIG=kozmic.config_local.DevelopmentConfig ./manage.py runserver`

4.1 Variables

SECRET_KEY A secret string. Used for signing cookie-based sessions, as a passphrase for private deploy keys, etc.

SERVER_NAME The name and port number of the server (e.g., `'kozmic-ci.company.com'` or `'127.0.0.1:5000'`).

SESSION_COOKIE_DOMAIN The domain for the session cookie. If this is not set, the cookie will be valid for all subdomains of *`SERVER_NAME`*.

> **Note:** If you're using an IP address as a *`SERVER_NAME`*, you must specify the same IP address in *`SESSION_COOKIE_DOMAIN`*. Otherwise cookies will not work.

KOZMIC_GITHUB_CLIENT_ID GitHub OAuth app client id

KOZMIC_GITHUB_CLIENT_SECRET GitHub OAuth app client secret

BROKER_URL Celery broker URL (default: `'redis://localhost:6379/0'`)

MAIL_DEFAULT_SENDER "From" e-mail address to be used for notifications

KOZMIC_REDIS_HOST Redis host (default: `'localhost'`)

KOZMIC_REDIS_PORT Redis port (default: `6379`)

KOZMIC_REDIS_DATABASE Redis database (default: `0`)

KOZMIC_STALL_TIMEOUT Number of seconds since the last job output after which the job is considered "hung" and it's Docker container gets killed (default: `900`)

KOZMIC_ENABLE_EMAIL_NOTIFICATIONS Whether e-mail notification enabled? (default: `True`)

KOZMIC_CACHED_IMAGES_LIMIT The maximum number of cached Docker images (a cached image is a result of an install script) per project (default: `3`)

KOZMIC_USE_HTTPS_FOR_BADGES If you're planning to use Kozmic CI status images in GitHub README files, they must be served through HTTPS to prevent GitHub from caching them.

This variable only affects the UI and used for showing a correct badge URL (default: `False`)

`SQLALCHEMY_DATABASE_URI` SQLAlchemy connection string (default: `'mysql+pymysql://kozmic:@127.0.0.1/kozm`

`TAILER_URL_TEMPLATE` URL template to be used to get a websocket URL for a job. Must point to a `tailer` application instance and contain `job_id` variable. (e.g., `'ws://kozmic-ci.example.com:8080/{job_id}/'`);

`DOCKER_URL` Docker API URL (default: `'unix://var/run/docker.sock'`)

The default configuration expects to find an SMTP server on a local machine on port 25. It can be changed: http://pythonhosted.org/Flask-Mail/#configuring-flask-mail.

CHAPTER 5

Internals Reference

5.1 Core

5.1.1 kozmic.models

class `kozmic.models.`**`RepositoryBase`**

A base repository class to be used by *`HasRepositories`* mixin.

`gh_id` **= Column(None, Integer(), table=None, nullable=False)**

GitHub id

`gh_name` **= Column(None, String(length=200), table=None, nullable=False)**

GitHub name (i.e., kozmic)

`gh_full_name` **= Column(None, String(length=200), table=None, nullable=False)**

GitHub full name (i.e., aromanovich/kozmic)

`gh_ssh_clone_url` **= Column(None, String(length=200), table=None, nullable=False)**

SSH clone url

`gh_https_clone_url` **= Column(None, String(length=200), table=None, nullable=False)**

HTTPS clone url

`is_public` **= Column(None, Boolean(), table=None, nullable=False)**

Is the repository public?

classmethod **`from_gh_repo`**`(`*gh_repo*`)`

Constructs an instance of *cls* from *gh_repo*.

class `kozmic.models.`**`HasRepositories`**

Mixin that adds `repositories` relationship to the model. Repositories are stored in separate tables for each parent. `Repository` attribute contains model (inherited from *`RepositoryBase`*) mapped to the parent's repositories table.

This pattern is well described in "Hand Coded Applications with SQLAlchemy" presentation by Mike Bayer.

class `kozmic.models.`**`User`**`(`***kwargs*`)`

User account.

`repositories`

Set of user repositories.

`organizations`

Set of user organizations in which user has admin rights to at least one repository (see *`Organization`*).

gh_id
GitHub user id

gh_login
GitHub user login

gh_name
Human-readable GitHub name

gh_access_token
OAuth access token

gh_avatar_url
GitHub avatar URL

repos_last_synchronized_at
The last time when the user's repositories and organizations were synced with GitHub

email
E-mail address

get_identity()
Returns user's `flask.ext.principal.Identity`.

get_available_projects(*annotate_with_latest_builds=False*)
Returns list of `Projects` that user has access to. If *annotate_with_latest_builds* is specified, returns list of pairs (`Projects`, *`Build`*) where the second element is the latest project build or `None` if the project was never built.

gh
An authenticated GitHub session for this user.

Type `github3.github.GitHub`

get_gh_org_repos()
Retrieves data from GitHub API and returns a pair of values:

1. `set` of `github3.orgs.Organization` in which the current user has at least one repository with admin rights;
2. `dict` mapping these organization' ids to lists of `github3.repo.Repository` to which the current user has admin access.

get_gh_repos()
Retrieves data from GitHub API and returns a list of the user owned repositories.

Return type list of `github3.repo.Repository`

sync_memberships_with_github()
Does the same as *`Project.sync_memberships_with_github()`*, but for the user. Returns True if there were not any GitHub errors; False otherwise.

class `kozmic.models.`**Organization**(***kwargs*)
Stores a set of organization repositories that a user has admin access to.

Different Kozmic users, but members of the same GitHub organization, will have their own *`Organization`* entries with possibly different sets of repositories (because they are possibly members of different teams).

repositories

gh_id
GitHub organization id

gh_login
GitHub organization login

gh_name
Human-readable GitHub name

user
`User` whose admin rights is reflected by this organization

class `kozmic.models.`**DeployKey**(*passphrase*, *key_size=2048*)
An RSA deploy key pair.

gh_id
GitHub deploy key id

rsa_private_key
RSA private deploy key in PEM format encrypted with the app secret key

rsa_public_key
RSA public deploy key in OpenSSH format

ensure()
If the corresponding GitHub deploy key does not exist, creates it. Returns True if there weren't GitHub API errors; False otherwise.

delete()
Deletes the public key from GitHub. Returns True if it has been successfully deleted (or was missing); False otherwise.

class `kozmic.models.`**Project**(***kwargs*)
Project is a GitHub repository that is being watched by Kozmic CI.

gh_id
GitHub repo id

gh_name
GitHub repo name (i.e., kozmic)

gh_full_name
GitHub repo full name (i.e., aromanovich/kozmic)

gh_login
GitHub repo owner (user or organization) login

gh_ssh_clone_url
SSH repo clone url

gh_https_clone_url
HTTPS clone url

is_public
Is the project's repository public?

deploy_key
Deploy key

members
Project members

owner
Project owner

gh
Project's GitHub.

Type `github3.repos.Repository`

delete()
Deletes the project and it's corresponding GitHub entities such as hooks, deploy key, etc. Returns True if they all have been successfully deleted (or were missing); False otherwise.

get_latest_build(*ref=None*)

Return type *Build*

sync_memberships_with_github()
Synchronizes project members with GitHub.

GitHub _repository **members_** with admin and push rights become project managers, other _repository **members_** become project members.

Returns True if there were not any GitHub errors; False otherwise.

class kozmic.models.**Hook**(***kwargs*)
Reflects a GitHub hook.

gh_id
GitHub hook id

title
Title

install_script
Install script

build_script
Script to be run at hook call

docker_image
Name of a Docker image to run build script in (for example, "ubuntu" or "aromanovich/ubuntu-kozmic"). Specified docker image is pulled from index.docker.io before build

project
Project

ensure()
If the corresponding GitHub hook does not exist, creates it. If it exists, but has wrong configuration, re-configures it. Returns True if there weren't GitHub API errors; False otherwise.

delete()
Deletes the project hook. Returns True if it's corresponding GitHub hook is missing or has been successfully deleted; False otherwise.

class kozmic.models.**TrackedFile**(***kwargs*)
Reflecs a tracked file.

path
Path within git repository

hook
Hook

class kozmic.models.**Build**(***kwargs*)
Reflects a project commit that triggered a project hook.

number
Build number (within a project)

gh_commit_ref
Commit reference (branch on which the commit was pushed)

gh_commit_sha
Commit SHA

gh_commit_author
Commit author

gh_commit_message
Commit message

created_at
Created at

status
Build status, one of the following strings: 'enqueued', 'success', 'pending', 'failure', 'error'

project
Project

calculate_number()
Computes and sets *number*.

started_at
Time the first job has started or None if there is no started jobs yet.

finished_at
Time the last job has finished or None if there is no finished jobs yet.

set_status(*status*, *target_url=''*, *description=''*)
Sets *status* and posts it on GitHub.

class kozmic.models.**HookCall**(***kwargs*)
Reflects a fact that GitHub triggered a project hook.

created_at
Created at

gh_payload
JSON payload from a GitHub webhook request

hook
Hook

build
Build

class kozmic.models.**Job**(***kwargs*)
A job that caused by a hook call.

started_at
Time the job has started or None

finished_at
Time the job has finished or None

return_code
Return code

`stdout`
Job log

`task_uuid`
uuid of a Celery task that is running a job

`build`
`Build`

`hook_call`
`HookCall`

`get_cache_id``()`
Returns a string that can be used for tagging a Docker image built from the install script. A cache id changes whenever the base Docker image, the install script or any of the tracked files is changed.

Note: Requires that Docker is running and Docker base image (`self.hook_call.hook.docker_image`) is pulled.

`started``()`
Sets *`started_at`* and updates *`build`* status. **Must** be called when the job is started.

`finished``(`*return_code*`)`
Sets *`finished_at`* and updates *`build`* status. **Must** be called when the job is finished.

`tailer_url`
URL of a websocket that streams a job log in realtime.

`permanent_url`
A permanent URL of the job.

`is_finished``()`
Is the job finished?

`status`
One of the following values: 'enqueued', 'success', 'pending', 'failure', 'error'.

5.1.2 kozmic.perms

`kozmic.perms.`**`project_owner`** **= <functools.partial object>**
Project owner need

`kozmic.perms.`**`project_manager`** **= <functools.partial object>**
Project manager need

`kozmic.perms.`**`project_member`** **= <functools.partial object>**
Project member need

`kozmic.perms.`**`delete_project`**`(`*id*`)`
Returns a `Permission` to delete the project identified by `id`.

`kozmic.perms.`**`manage_project`**`(`*id*`)`
Returns a `Permission` to manage the project identified by `id`.

`kozmic.perms.`**`view_project`**`(`*id*`)`
Returns a `Permission` to view the project identified by `id`.

5.1.3 kozmic.builds.tasks

`kozmic.builds.tasks.`**`do_job`**(*hook_call_id*)

A Celery task that does a job specified by a hook call.

Creates a `Job` instance and executes a build script prescribed by a triggered `Hook`. Also sends job output to `Job.task_uuid` Redis pub-sub channel and updates build status.

Parameters **`hook_call_id`** – int, `HookCall` identifier

`kozmic.builds.tasks.`**`restart_job`**(*id*)

A Celery task that restarts a job.

Parameters **`id`** – int, `Job` identifier

class `kozmic.builds.tasks.`**`Publisher`**(*redis_client*, *channel*)

Parameters

- **`redis_client`** – Redis client
- **`channel`** (*`str`*) – pub/sub channel name

class `kozmic.builds.tasks.`**`Tailer`**(*log_path*, *publisher*, *container*, *kill_timeout=600*)

A daemon thread that waits for additional lines to be appended to a specified log file. Once there is a new line, it does the following:

1.Translates ANSI sequences to HTML tags;

2.Sends the line to a Redis pub/sub channel;

3.Pushes it to Redis list of the same name.

If the log file does not change for `kill_timeout` seconds, specified Docker container will be killed and corresponding message will be appended to the log file.

Once the thread has finished, `has_killed_container` tells whether the **:param:‘container‘** has stopped by itself or been killed by a timeout.

Parameters

- **`log_path`** (*`str`*) – path to the log file to watch
- **`publisher`** (*`Publisher`*) – publisher
- **`container`** (dictionary returned by `docker.Client.create_container()`) – container to kill
- **`kill_timeout`** (*`int`*) – number of seconds since the last log append after which kill the container

class `kozmic.builds.tasks.`**`Builder`**(*docker*, *message_queue*, *docker_image*, *script*, *working_dir*, *clone_url*, *commit_sha*, *deploy_key=None*)

A thread that starts a script in a container and waits for it to complete.

One of the following attributes is not `None` once the thread has finished:

`return_code`

Integer, a build script's return code if everything went well.

`exc_info`

`exc_info` triple `(type, value, traceback)` if something went wrong.

Parameters

- **`docker`** (`docker.Client`) – Docker client

- **message_queue** (`Queue.Queue`) – a queue to which put an identifier of the started Docker container. Identifier is a dictionary returned by `docker.Client.create_container()`. Builder will block until the message is acknowledged by calling `Queue.Queue.task_done()`.
- **deploy_key** (*2-tuple of strings*) – a pair of strings (private key, passphrase)
- **docker_image** (*str*) – a name of Docker image to be used for `build_script` execution. The image has to be already pulled from the registry.
- **working_dir** (*str*) – path of the directory to be mounted in container's */kozmic* path
- **clone_url** (*str*) – SSH clone URL
- **commit_sha** (*str*) – SHA of the commit to be checked out

5.2 Packages Overview

5.2.1 kozmic

`kozmic.__init__.`**`create_app`**(*config=None*)

Returns a fully configured `Flask` application.

Parameters config – a config object or it's name. Will be passed directly to `flask.config.Config.from_object()`. If not specified, the value of `KOZMIC_CONFIG` environment variable will be used. If `KOZMIC_CONFIG` is not specified, `'kozmic.config.DefaultConfig'` will be used.

5.2.2 kozmic.accounts

`kozmic.accounts.`**`bp`**

`flask.Blueprint` that gives users a means to manage their account settings.

5.2.3 kozmic.auth

`kozmic.auth.`**`bp`**

`flask.Blueprint` that implements an authentication through GitHub.

5.2.4 kozmic.builds

`kozmic.builds.`**`bp`**

`flask.Blueprint` that implements webhooks to be triggered by GitHub and serves status badges.

Note: Does not require authentication.

5.2.5 kozmic.projects

`kozmic.projects.`**`bp`**

`flask.Blueprint` that provides all the means for managing and viewing projects.

5.2.6 kozmic.repos

`kozmic.repos.`**`bp`**

`flask.Blueprint` that gives users the abilities to:

1. View list of GitHub repositories they have admin access to
2. Create *`kozmic.models.Project`* for any of them

CHAPTER 6

Contributing

This document is far from extensive, but hopefully it gives an idea of how to deploy a development version of Kozmic CI and get started.

- Clone the source code from GitHub repository: https://github.com/aromanovich/kozmic-ci
- Install the Python dependencies using pip:

```
pip install -r requirements/kozmic.txt
pip install -r requirements/tailer.txt
pip install -r requirements/dev.txt
```

- Take a look at *Configuration* variables and fill the configuration file `kozmic/config_local.py` using `kozmic/config_local.py-dist` as an example.

6.1 Running the components

- Run the development server:

```
KOZMIC_CONFIG=kozmic.config_local.DevelopmentConfig ./manage.py runserver
```

- Run the Celery worker:

```
KOZMIC_CONFIG=kozmic.config_local.DevelopmentConfig \
celery worker -A kozmic.entry_point.celery -l debug
```

- Run the tailer component:

```
KOZMIC_CONFIG=kozmic.config_local.DevelopmentConfig \
uwsgi --http-socket :8080 --gevent 5 --gevent-monkey-patch -H ~/Envs/kozmic/ \
      --module tailer:app
```

  Note that tailer app has to be run using uWSGI that is listed in `requirements/tailer.txt`. If you use a virtual environment (which is strongly advised), path to it must be specified using `-H` argument.

6.2 Running tests

- Run all tests: `./test.sh`
- Run tests that don't require Docker: `./test.sh -m "not docker"`
- Run the particular test: `./test.sh -k TestUserDB`

6.3 Working with the database

`./manage.py db` provides an interface to Alembic, a database migration tool. Run `./manage.py db --help` to figure out what commands it has. The most useful are:

- Apply database migrations:

```
    KOZMIC_CONFIG=kozmic.config_local.DevelopmentConfig ./manage.py db upgrade
```

- Automatically generate a new migration:

```
    KOZMIC_CONFIG=kozmic.config_local.DevelopmentConfig ./manage.py db migrate
```

6.4 Compiling the documentation

```
cd docs
KOZMIC_CONFIG=kozmic.config_local.DevelopmentConfig make html
```

CHAPTER 7

Changelog

7.1 0.0.1: 18 May 2014

The first public release.

CHAPTER 8

Contact

- Issues on GitHub: https://github.com/aromanovich/kozmic-ci/issues
- Twitter: @antonromanovich
- Private email: anthony.romanovich [at] gmail