Docker Project Template Documentation

Release 2

Wolnosciowiec Team

Jul 21, 2020

Contents:

docker-compose based framework for building production-like environments - developing and testing on your local computer, deploying to your server or cluster from shell or from CI. **Harbor is a pre-configured set of most popular technologies available to use with docker-compose, in addition of our exclusive features**

Features:

- Service discovery (pins containers into WWW domains by labelling)
- Deployment strategies: compose's standard, recreation, and **rolling-updates (zero-downtime updates)**
- Automatic Letsencrypt SSL
- Standardized directory structures and design patterns
- Ready to use snippets of code and solutions
- Ansible integration to prepare your production/testing server and deploy updates in extremely intuitive way

CHAPTER 1

Getting started

1. Install Harbor

```
pip install rkd-harbor
```

2. Create GIT project

```
mkdir my-project
cd my-project

git init
```

3. Create Harbor project

```
harbor :create:project
```

1.1 Read about project Structure

CHAPTER 2

Structure

Project consists of a standard structure which includes:

2.1 .rkd

RiotKit-Do directory, where you can define custom tasks, there are also temporary files and logs stored (ADVANCED)

2.2 apps/conf/

docker-compose YAML files with definitions of containers, networks and volumes

2.3 apps/conf.dev/

Same as apps/conf, but enabled only on development environment

2.4 apps/profile/

Defined service profiles that allows to select services on which you operate in given command (eg. wordpress profile = all instances of wordpress)

2.5 apps/healthchecks/

RiotKit's InfraCheck integration, here are placed all of the healthcheck definitions (see section about health and monitoring)

2.6 apps/repos-enabled/

GIT repositories definitions (see section about applications from external GIT repositories)

2.7 apps/www-data/

Cloned applications from GIT (see section about applications from external GIT repositories)

2.8 containers/

Configuration data mounted via bind-mount to inside containers (should be read-only and versioned by GIT)

2.9 data/

Bind-mounted volume storage for containers, only data that is generated dynamically by containers is stored there.

Example use cases:

- Database data eg. /var/lib/mysql
- Generated SSL certificates storage
- NGINX generated configurations

2.10 hooks.d/

Scripts that are executed at given time in the Harbor lifecycle (eg. pre-start, post-start, pre-upgrade, . . .) See section about hooks.

```
hooks.d/
hooks.d/pre-upgrade
hooks.d/(...)
hooks.d/post-start
```

2.10.1 Keeping standards

KISS - keep it simple stupid

By keeping standards in your project you make sure, that any person that joins your project or a contributor could be satisfied with Harbor documentation. Any outstanding solutions would require you to create extra documentation in your project.

2.10.2 OK, got it, let's learn Basic commands

CHAPTER 3

Basic commands

Harbor defines a lot of RKD tasks, that automates preparing changes to project as well as operating on live organism.

At first we would like you to get familiar with our Snippet Cooperative, which is a place to share a code with others - for you now it means you can install an application with a single command.

3.1 Installing applications from Snippet Cooperative

Browse the catalogue of applications at: https://github.com/riotkit-org/harbor-snippet-cooperative Then, perform an installation within a single command.

```
# at first do a repository sync, later you don't need to do it all the time
harbor :cooperative:sync :cooperative:install harbor/redis
```

3.2 Starting, upgrading and stopping services

Basic tasks lets you control the services running in your environment, just like you were doing before with **docker ps** but with a difference that Harbor interface is more domain-focused interface.

```
# create and start containers
harbor :start

# pull new images, update git repositories, then start
harbor :upgrade

# start selected service
harbor :service:up hello

# remove selected service
harbor :service:rm hello

```

(continues on next page)

(continued from previous page)

```
# list all services, running and not running
harbor :service:list

# check a report of running service
harbor :service:report hello
```

3.3 Controlling maintenance mode on production

Sometimes, when bad things happens, or a scheduled repair is planned a maintenance mode is required. Harbor provides a simple maintenance mode in 3 ways: global, per-service, per-domain

```
# maintenance mode per single service
harbor :maintenance:on --service hello

# per single domain
harbor :maintenance:on --domain domain-name.org

# global maintenance mode - for all services
harbor :maintenance:on --global
```

3.4 Diagnosing issues (advanced usage on production)

# do the docker-compose ps, in case you need harbor :diagnostic:compose:ps

# in case you need a full docker-compose arguments used by Harbor to execute some commands manually harbor :diagnostic:dump-compose-args

# dump all yamls to big one for analysis harbor :diagnostic:compose:config

# force regenerate all Letsencrypt certificates (use with caution, there are limits of hits on Letsencrypt) harbor :gateway:ssl:regenerate

# reload gateway in case, when the the nginx.tmpl was modified harbor :gateway:reload

CHAPTER 4

Creating first service

Service definitions are docker-compose.yml files, with addition of Harbor's patterns which allows to automate and standardize the way of environment preparation.

Few rules:

- YAML files are stored at `./apps/conf`
- The naming: `apps.MY-APP-NAME.yaml` for applications, and `infrastructure.MY-TECHNICAL-APP-NAME.yaml` for technical services (health checks, backups etc.)
- Volumes with configuration files eg. nginx.conf - should be in `./container/MY-APP-NAME` directory
- Volumes with external git repositories should be in `./apps/www-data/MY-APP-NAME` directory
- Volumes with dynamic data such as user uploads should be in `./data/MY-APP-NAME` directory

4.1 Let's create first service then!

Best way to create a service is to use a generator - to avoid common mistakes.

Demo: https://asciinema.org/a/348867

```
harbor :cooperative:sync
harbor :cooperative:install harbor/webservice
```

The below example will sync coop repositories, then use `harbor/webservice` template to generate docker-compose yaml file, that will be placed in `./apps/conf` directory.

Use `:service:up` task to bring up a recently created service.

```
harbor :service:list
harbor :service:up service-name
```

After checking that everything works correctly the service definition + configuration files placed in `./container` directory should be pushed to GIT.

CHAPTER 5

From authors

Project was started as a part of RiotKit initiative, for the needs of grassroot organizations such as:

- Fighting for better working conditions syndicalist (International Workers Association for example)
- Tenants rights organizations
- Various grassroot organizations that are helping people to organize themselves without authority

RiotKit Collective