Docker Project Template Documentation

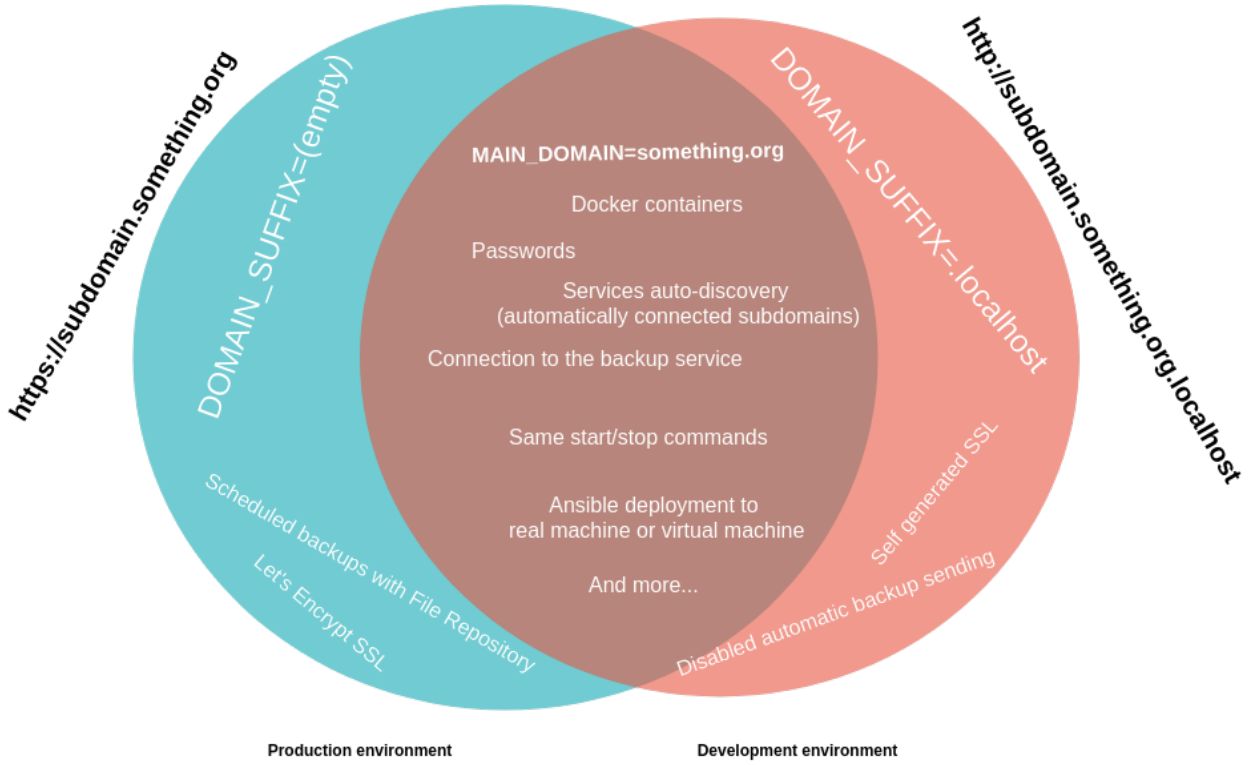
Release 2

Wolnosciowiec Team

May 02, 2019

Contents:

Comprehensive single-server docker deployment template. Perfect for smaller and medium projects. Unified production and development environment, with minimum amount of differences.



Includes:

- **Service discovery, automatic SSL (generates NGINX configuration on-fly for domains, subdomains + SSL)**
- Support for webhooks
- Ansible integration (ready to use role)
- **Encrypted production credentials (.env-prod)**
- **Modularity, template is split into parts that could be enabled/disabled**
- YAML based configuration, clear and easy to maintain
- **Health checks integration** + simple dashboard
- Services index (to publish list of installed apps for non-technical users)
- **Automatic backups** to external server (File Repository integration)
- Ready-to-use SMTP relay, easy to configure
- **Support for git-based projects mounted as volumes**
- Updater to keep your template up-to-date with RiotKit's Harbor
- Templating system for generating configuration files
- Database migrations
- **Maintenance mode**

Goals:

- Provide complete, automated infrastructure
- Easy of use and easy to understand
- Feature toggle on/off
- Template for common usages
- Integration with other RiotKit's projects such as health checking, automated backups

CHAPTER 1

First steps

Naturally you need **docker**, **docker-compose** on a Linux-based host OS, basic understanding of how the docker containers works and at least little Linux shell experience.

To install:

- git (repository)
- docker (get.docker.com)
- docker-compose (get.docker.com)

1.1 Let's go!

Project template is hosted on **git** version control system, with git it's possible to update your environment with newer template version. Updates are not mandatory, and the template may be not always backwards compatible, because it's a template you can use fully or only parts of it.

1. Create your project directory (replace "your-project-name" with a proper name):

```
mkdir your-project-dir
cd your-project-dir

# initialize git repository, at least locally
git init

# download the project files using updater script
curl -s https://raw.githubusercontent.com/riotkit-org/riotkit-harbor/master/update-
↪from-template.sh | bash
```

2. Take a look around, check documentation for:

- *Project structure*
- *Features*

- *Configuration reference*

You may be interested with placing your docker container definitions at **./apps/conf**

3. Configure the project:

Before you will start changing the configuration, please a look at the *Configuration concept*.

```
cp .env-default .env
edit .env
```

4. Start the project:

```
make help
make start
```

5. Access configured domain on web browser

Go to http://the-configured-domain-here.localhost, enjoy.

You may want to check a complete example: *General conception*

CHAPTER 2

Project structure

```
- apps
    - conf
    - continuous-deployment
    - healthchecks
    - repos-enabled
    - www-data
- containers
- data
- hooks.d
```

2.1 Apps

Most important section, here you put application definitions, configure healthchecks for monitoring, repositories for versioning of applications in volumes.

conf:

```
docker-compose files.
".disabled" suffix disables a file.

To enable/disable use "make config_enable APP_NAME=smtp"
To list enabled: "make  make list_configs_enabled"
Listing disabled: "make list_configs_disabled"
```

continuous-deployment:

Allows to configure a thin-deployer which can take incoming webhooks and turn into the commands. Example scenario: When a push to docker registry happens, then a HTTP request is made, thin-deployer runs a command that updates the container.

healthchecks:

Status monitoring, in "checks" you can add custom health check functions (scripts in any language), in "configured" you define usage and parameters. The environment uses infracheck application for healthchecking.

repos-enabled and www-data:

If you have applications that are mounted as volume in a generic container, you may manage them and deploy new versions using the application-as-volume feature.

2.2 Containers

The containers directory contains data that is added to containers as a volume. The difference from "data" directory is, that "containers" is kept in git, it's a place for configuration files. "data" is mostly the binary data generated by applications.

2.3 Data

Binary data generated by containers, eg. MySQL tables, logs. Mounted as volume to applications. Not kept in git.

2.4 Hooks.d

A place to keep scripts that launch before environment starts, stops etc.

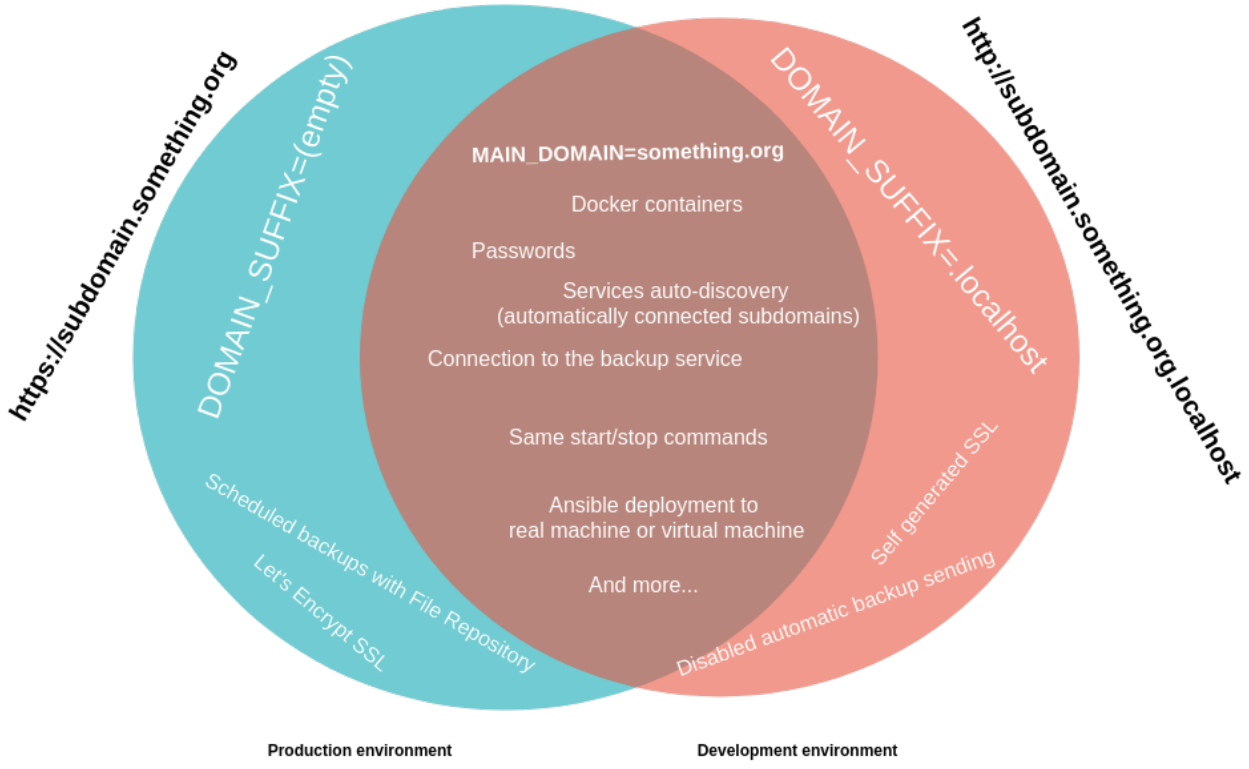
```
- deployment-pre: Runs on "make deployment_pre", used by Ansible role
- post-down: Runs when environment stops
- post-start: Runs right after "make start"
```

CHAPTER 3

General conception

The idea of this environment template is to provide a base for small and medium projects, deployed on a single server. With a high focus on ease of use. The environment should be also almost the same on development server as on production server.

There are a few **design patterns**, that are the basis of the environment conception.



3.1 Management from Makefile

Everything important goes into the Makefile. There are no plain bash scripts outside of the Makefile. The project is built in purely Makefile + YAML + docs + misc files.

```
make start
make check_status
make list_all_hosts
make stop
make build_docs
make encrypt_env_prod

# ...
# and others
```

3.2 Multiple compose files can be toggled on/off

Service definitions in Docker Compose format are kept in **./apps/conf** directory. Services that are temporarily disabled are marked with ".disabled" at the end of filename.

```
 make list_configs
dashboard
deployer
health
service-discovery
smtp
ssl
technical
uptimeboard

 make config_disable APP_NAME=ssl
 >> Use APP_NAME eg. make config_disable APP_NAME=iwa-ait
 OK, ssl was disabled.
 make config_enable APP_NAME=ssl
 >> Use APP_NAME eg. make config_disable APP_NAME=iwa-ait
 OK, ssl is now enabled.
```

3.3 Configuration in one file that could be encrypted

Good practice is to extract environment variables into .env files, instead of hard-coding values into services YAML definitions. That makes a **.env** file from which we can use environment variables in YAML files with syntax eg. ${VAR_NAME}

As the **.env** cannot be pushed into the repository, there is a possibility to push **.env-prod** as a encrypted file with ansible-vault.

```
make encrypt_env_prod
```

3.4 Main domain and domain suffix concept

MAIN_DOMAIN can be defined in **.env** and reused in YAML files together with **DOMAIN_SUFFIX**. It opens huge possibility of creating test environments, which have different DNS settings. Sounds like a theory? Let's see a practical example!

Scenario for test environment:

```
Given It's a TEST environment
So the variables are configured in following way
| DOMAIN_SUFFIX | .localhost  |
| MAIN_DOMAIN   | iwa-ait.org |
When application has set VIRTUAL_HOST=some-service.${MAIN_DOMAIN}${DOMAIN_SUFFIX}
Then the SOME SERVICE will have address http://some-service.iwa-ait.org.localhost
```

Scenario for production environment:

```
Given It's a TEST environment
So the variables are configured in following way
| DOMAIN_SUFFIX |             |
| MAIN_DOMAIN   | iwa-ait.org |
When application has set VIRTUAL_HOST=some-service.${MAIN_DOMAIN}${DOMAIN_SUFFIX}
Then the SOME SERVICE will have address http://some-service.iwa-ait.org
```

It's so much flexible that you can host multiple subdomains on main domain, but you can also use totally different domains.

Example:

```
MAIN_DOMAIN=iwa-ait.org
DOMAIN_SUFFIX=.localhost
```

```
first:
    environment:
        - VIRTUAL_HOST=some-service.${MAIN_DOMAIN}${DOMAIN_SUFFIX}

second:
    environment:
        - VIRTUAL_HOST=other-service.example.org${DOMAIN_SUFFIX}
```

In result of above example you will have services under domains in test environment:

- some-service.iwa-ait.org.localhost
- other-service.example.org.localhost

Complete example

In *.env* file:

```
MAIN_DOMAIN=iwa-ait.org
DOMAIN_SUFFIX=.localhost
```

In *./apps/conf/docker-compose.phpmyadmin.yaml*:

```
db_mysql_admin:
    image: phpmyadmin/phpmyadmin
    environment:
```

(continues on next page)

(continued from previous page)

```
      - PMA_HOST=db_mysql

      # gateway configuration
      - VIRTUAL_HOST=pma.${MAIN_DOMAIN}${DOMAIN_SUFFIX}
      - VIRTUAL_PORT=80
  labels:
      org.riotkit.dashboard.enabled: true
      org.riotkit.dashboard.description: 'MySQL database management'
      org.riotkit.dashboard.icon: 'pe-7s-server'
      org.riotkit.dashboard.only_for_admin: true
```

Now you can access http://pma.iwa-ait.org.localhost in your browser. On production server just remove the DOMAIN_SUFFIX value to have http://pma.iwa-ait.org - simple enough, huh?

3.5 Automatic distinction between development and production server

There should be no need to have separated configuration files for local development environment, and for production environment. Everything should be REALLY the same, except **DOMAIN_SUFFIX** variable, which should point to **.localhost** on development environment.

Whenever you will need to pass information to some docker container, that we are in **debug mode** you can use **${IS_DEBUG_ENVIRONMENT}** in YAML definition. **IS_DEBUG_ENVIRONMENT** is a result of autodetection if the environment is local or production, you may also set **ENFORCE_DEBUG_ENVIRONMENT=1** if you want to enforce debug environment.

HINT: File Repository's Bahub integration configuration integrates with IS_DEBUG_ENVIRONMENT by stopping cronjobs, no backups are done from developer environment HINT: Ansible deployment is able to modify .env variables when pushing changes to production.

3.6 Applications pulled from git repositories

Not always it's possible to package an application into container. If we have a private application without public source code, and we do not have a private docker registry - then it's possible to use a generic eg. PHP + NGINX container and **mount the application files as a volume**.

CHAPTER 4

Configuration concept

The configuration is placed at the **.env** file, which is passed to docker-compose configuration. Services see only a few variables, or all variables (it depends if "environment" or "env_file" section was used in service configuration). A **.env-default** file is a template containing example, default values.

```
.env
.env-default
.env-prod
```

Ansible deployment is using **.env-prod** when pushing application to the server, decrypting it and replacing **.env** file on the server.

> *NOTE: .env is not to store in the git. To store production credentials please use .env-prod encrypted with ansible-vault and decrypted during Ansible deployment*
>
> *NOTE: .env-default should not contain passwords as it is stored in git*

4.1 Test and production environment

They are mostly the same except fact, that the domains are different and, the test environment does not have real SSL enabled.

Domain suffix design pattern, is a pattern where we have a BASE DOMAIN eg. riotkit.org, in test environment we add a suffix, so it would be riotkit.org.localhost on our test environment. This practice gives us out-of-the-box working DNS on local testing machine.

Example production specific .env part:

```
DOMAIN_SUFFIX=
MAIN_DOMAIN=riotkit.org
```

Example test specific .env part:

```
DOMAIN_SUFFIX=.localhost
MAIN_DOMAIN=riotkit.org
```

CHAPTER 5

Features

5.1 Service discovery with SSL

Whenever you run a new application, even without restarting the whole environment, the **gateway_proxy_gen** container is being notified about it and creates an nginx cnofiguration to expose your application. **gateway_letsencrypt** is also notified and generates a SSL configuration, right after nginx is ready.

Step-by-step schema:

1. **gateway_proxy_gen** listens to docker events
2. **gateway_proxy_gen** generates nginx.conf for each TAGGED service in docker network (VIRTUAL_HOST, VIRTUAL_PORT, LETSENCRYPT_HOST and LETSENCRYPT_EMAIL must be present)
3. **gateway_letsencrypt** generates SSL certificates for recently added domains
4. **gateway** is being reloaded and uses new nginx.conf with SSL certificates

FAQ:

1. Multiple domains: Separate them by comma
2. For advanced usages see: docker-gen documentation
3. I need to modify nginx.conf, how to? It's easy. There are two solutions:

   a) Create a file with domain name in the **./containers/nginx/vhost.d** directory

   Example "test.mydomain.org" file:

```
## Start of configuration add by letsencrypt container
location ^~ /.well-known/acme-challenge/ {
    auth_basic off;
    allow all;
    root /usr/share/nginx/html;
    try_files $uri =404;
    break;
}
```

(continues on next page)

(continued from previous page)

```
## End of configuration add by letsencrypt container

proxy_read_timeout 120s;
```

b) Modify directly the template from which the nginx.conf is created

NOTE: During the make start the docker-gen container is modifying files in vhost.d directory. On production a script reset-nginx-conf-permissions.sh in make stop will attempt to restore file contents to allow git pull execution later

5.2 Services dashboard

Often non-technical people are not aware of what services are actually hosted. There we resolve this problem with an automatically generated list of running web-apps.

Applications needs to be tagged with docker labels, example:

```
org.riotkit.dashboard.enabled: true
org.riotkit.dashboard.description: 'Dashboard - a list of all hosted websites running␣
↪on this network'
org.riotkit.dashboard.icon: 'pe-7s-browser'
org.riotkit.dashboard.only_for_admin: false
```

5.3 Sending e-mails

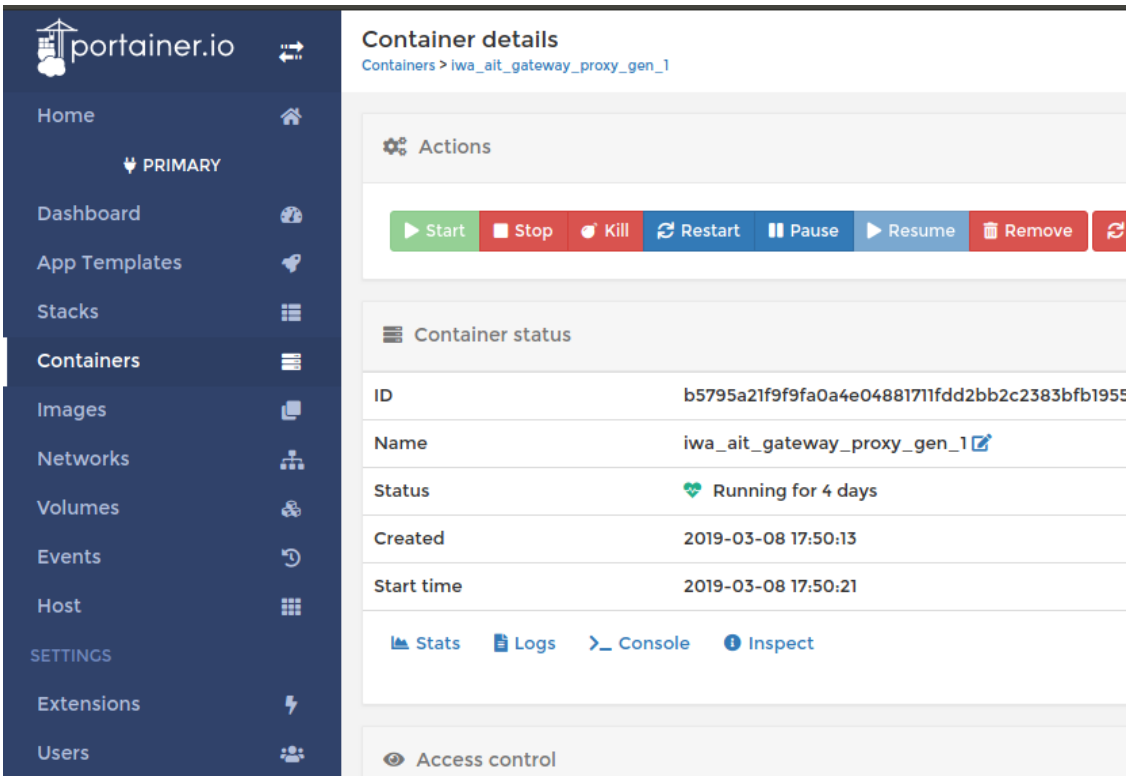
Send e-mails directly or through a middle server, just by sending them on "smtp_server:25" without any authorization. You can use any external SMTP, your own, a gmail account, or other.

```
SMARTHOST_ADDRESS=your-server.org     # your SMTP relay server address
SMARTHOST_PORT=587
SMARTHOST_USER=login@your-server.org
SMARTHOST_PASSWORD=
SMARTHOST_ALIASES=*     # forward all e-mails, you can put here eg. allowed recipient␣
↪domains
```

If you make values empty, then the service will send mails directly.

5.4 Docker administration panel

We suggest to optionally use *Portainer* in case a need for web access to the environment. *Portainer* is very lightweight and has support for almost all important features.

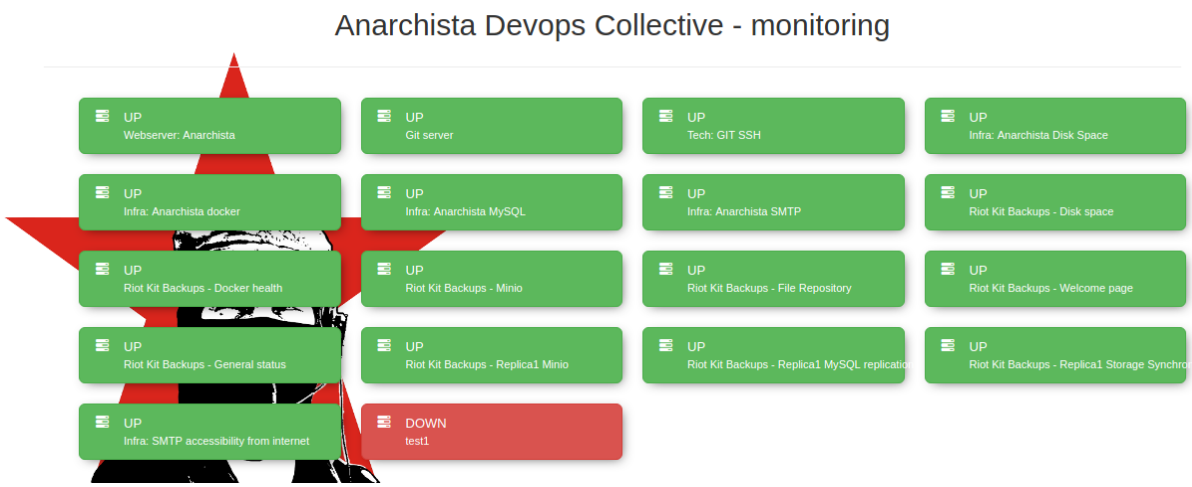


5.5 Uptime Board

Dashboard with health check status. The status is fetched from an external service provider.

Configuration in .env file:

```
MONITORING_PROVIDERS=UptimeRobot://some-token;UptimeRobot://some-other-token
```



5.6 Integration with Ansible

See: *Ansible deployment*

5.7 Templating

Imagine that everything is in environment variables in **.env**, but you need to eg. initialize the MySQL database by creating multiple users and databases. To allow keeping passwords safe in the **.env-prod**, but still automating eg. the user and password creation in databases we can use a templating mechanism.

```
Templating is GENERATING configuration files from templates, while having access to .
↪env variables.
```

Workflow:

1. Create a template that will be rendered on **make start**
2. Mount compiled template to a volume eg. to the entrypoint.d directory

Example ./containers/templates/source/mysql/access.sql.j2

```
/* transprzyjazn.pl */
CREATE DATABASE IF NOT EXISTS transprzyjazn;
CREATE USER IF NOT EXISTS 'transprzyjazn'@'%' IDENTIFIED BY '{{ DB_PASSWD_
↪TRANSPRZYJAZN }}';
GRANT ALL ON `transprzyjazn`.* TO 'transprzyjazn'@'%' IDENTIFIED BY '{{ DB_PASSWD_
↪TRANSPRZYJAZN }}';

/* lokatorzy.info.pl */
CREATE DATABASE IF NOT EXISTS lokatorzy;
CREATE USER IF NOT EXISTS 'lokatorzy'@'%' IDENTIFIED BY '{{ DB_PASSWD_LOKATORZY_INFO_
↪PL }}';
GRANT ALL ON `lokatorzy`.* TO 'lokatorzy'@'%' IDENTIFIED BY '{{ DB_PASSWD_LOKATORZY_
↪INFO_PL }}';
```

You need to define environment variables in the **.env** (on following example: DB_PASSWD_TRANSPRZYJAZN and DB_PASSWD_LOKATORZY_INFO_PL). The file will be rendered into **./containers/templates/compiled/mysql/access.sql.j2** So, now you can mount the compiled mysql files directory into the MySQL container for example.

```
version: "2"

volumes:
    mysql-data:

services:
    #
    # MySQL server
    #
    #  In ./containers/templates/source/mysql put JINJA2 templates of SQL files, they will be compiled to ./containers/templates/compiled/mysql and run on MySQL start
    #  you can create users, databases with this method on MySQL startup, while keeping the passwords in the .env file
    #
    #  --innodb_file_per_table=1 is a secure setting to keep tables separated (in case of disk failure easier to recover data)
    #
    #  All MySQL databases data is stored in "./data/mysql". For backup configuration use "docker_volumes"
    #  (docker_hot_volumes not recommended) with path /var/lib/mysql
    #
    db:
        image: mariadb
        volumes:
            - "mysql-data:/var/lib/mysql"     # use a named volume
            #- "./data/mysql:/var/lib/mysql"  # use a bind-mount (possibly slower, but the files would be in one folder)
            - "./containers/templates/compiled/mysql:/docker-entrypoint-initdb.d/"
        command: mysqld --innodb_file_per_table=1
        environment:
            - MYSQL_USER=${MYSQL_USER}
            - MYSQL_PASSWORD=${MYSQL_PASSWORD}
            - MYSQL_DATABASE=${MYSQL_DATABASE}
            - MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD}
        labels:
            com.centurylinklabs.watchtower.enable: true


    #
    # MySQL admin panel
    #
    db_mysql_admin:
        image: phpmyadmin/phpmyadmin
        environment:
            - PMA_HOST=db

            # gateway configuration
            - VIRTUAL_HOST=dba.${MAIN_DOMAIN}${DOMAIN_SUFFIX}
            - VIRTUAL_PORT=80
            - LETSENCRYPT_HOST=dba.${MAIN_DOMAIN}${DOMAIN_SUFFIX}
            - LETSENCRYPT_EMAIL=${LETSENCRYPT_EMAIL}
        expose:
            - "80"
```

(continues on next page)

(continued from previous page)

```
        labels:
            com.centurylinklabs.watchtower.enable: true
```

5.8 Backups

See: *Guide to failure-safe server*

5.9 Automatic containers update

Watchtower keeps an eye on containers marked with *com.centurylinklabs.watchtower.enable* label. Each container's image is checked for update availability, if an update is available then it's pulled from registry and the container is re-created on a new version of image.

Downtime is minimized by pulling newer versions of images at first, then re-creating containers in proper order. Linked containers dependency chain is respected, so the containers are re-created in proper order.

To enable Watchtower, just use a template "infrastructure.updates.yml.example", copy it to the conf directory with removing ".example" suffix.

Configuration

By default there are a few example variables extracted into the environment. You may adjust it to your needs, turn off notifications, or switch notifications from slack/mattermost to e-mail.

Check Watchtower documentation for detail.

```
# watchtower
WATCHTOWER_INTERVAL=1800
WATCHTOWER_SLACK_HOOK=...
WATCHTOWER_IDENTIFIER="Watchtower"
```

5.10 Securing access with basic auth

Gateway can implement a basic auth standard for selected domains you want to secure with password.

A container needs to be marked with a label, example:

```
labels:
    org.riotkit.htaccessFile: "dashboard.htpasswd"
```

The "dashboard.htpasswd" file needs to be placed in **./containers/auth/** directory. To generate a password use this example command:

```
htpasswd -c ./containers/auth/dashboard.htpasswd admin
```

5.11 Maintenance mode

When there are technical issues you may possibly want to show a nice error page, instead of "could not connect to redis". To achieve this goal RiotKit's environment implements a **maintenance mode**.

How it works

- The gateway is checking if */maintenance/on* file exists, if yes, then displays a maintenance page for all domains/containers labelled with "org.riotkit.useMaintenanceMode: true"
- You can turn on/off manually maintenance mode with "make maintenance_on" and "make maintenance_off"
- You can turn it on/off AUTOMATICALLY using infracheck healthchecks, add file creation and deletion as hooks (infracheck executes all checks every one minute by default)

Docker label required to enable maintenance mode for selected container

```
org.riotkit.useMaintenanceMode: true
```

Toggle from shell

```
make maintenance_on
make maintenance_off
```

Automatic maintenance mode with Infracheck

Infracheck can execute checks each 1 minute (it is configurable via CHECK_INTERVAL), triggering hooks on success and failure. This means that we can turn on the maintenance mode, when for example MySQL will go down for a backup process.

```
{
    "type": "port-open",
    "input": {
        "po_port": "3306",
        "po_host": "db_mysql",
        "po_timeout": "1"
    },
    "hooks": {
        "on_each_down": [
            "touch /maintenance/on"
        ],
        "on_each_up": [
            "rm -f /maintenance/on"
        ]
    }
}
```

Modifying HTML templates

Everything is placed at *./containers/nginx/maintenance* and mounted as volume.

5.12 WWW to non-www redirection

To enable automatic redirection from www. to non-www domain you need to use a label and inform also letsencrypt about a subdomain.

```
environment:
    - VIRTUAL_HOST=aitrus.info${DOMAIN_SUFFIX}
    - LETSENCRYPT_HOST=aitrus.info${DOMAIN_SUFFIX},www.aitrus.info${DOMAIN_SUFFIX}
labels:
    org.riotkit.redirectFromWWW: true
```

CHAPTER 6

Ansible deployment

To have a complete, reproducible environment which includes host SSH, shell, automatic updates, users, groups, firewall, vpn an Ansible could be used. There exists an official ready-to-use role in separate repository.

6.1 What is Ansible?

Imagine that you do not have to log-in into your server to change it's configuration, there are easier ways. You define a set of YAML files with TASKS TO EXECUTE to have a given result.

Examples tasks:

- Create user X, group Y
- Install A, B, C packages
- Block all traffic, except port 80, 22
- **Install and configure your project that is based on Docker Project Template for example**

From Wikipedia:

```
Ansible is an open-source software provisioning, configuration management, and␣
↪application deployment tool
```

6.2 Deploying a project using this docker environment template

A role from this repository can be used simply, for example in this way:

```
- role: blackandred.server_docker_project
  tags: project
  vars:
      deploy_user: tech.admin
```

(continues on next page)

(continued from previous page)

```
      project_dir: /project
      git_deploy_url: "https://your-git-server/backup-replica-environment.git"
      git_regular_deploy_url: "https://your-git-server/backup-replica-environment.git"
      make_executable: "sudo ./make.sh"

      test_specific_env:
          - { line: "DOMAIN_SUFFIX=.localhost", regexp: '^DOMAIN_SUFFIX', title:
↪'env: Add domain suffix - .localhost' }

      production_specific_env:
          - { line: "DOMAIN_SUFFIX=", regexp: '^DOMAIN_SUFFIX', title: 'env: Remove␣
↪domain suffix' }
```

1. Add above to your playbook
2. Install role with *ansible galaxy install blackandred.server_docker_project*
3. Run *ansible-playbook your-playbook.yml -i hosts.cfg -t project*

6.3 Storing credentials

You will probably have a dilemma when it will go into the point to have some passwords on the production server. The passwords should not be stored in git repository as plain text, so *.env* and *.env-default* should not be used.

There is a solution.

1. Just put everything into *.env* file locally, by default its ignored from sending to git (.gitignore), do not commit it.
2. Create *".vault-password"* file in root directory of your project repository, make sure it's in .gitignore so it will be not placed in repository
3. Optional: You can keep *".vault-password"* in safe place, eg. on your pendrive linked symbolically to the project directory
4. Use *make encrypt_env_prod* to generate *.env-prod* that will be used by Ansible to create *.env* on production server during deployment

6.4 Limited downtime deployments

Docker Compose has an ability to detect which service YAML file was changed and recreate only containers for changed services. This does not apply to files mounted via volumes.

If you made a change to one of files mounted via volumes, then you need to restart the environment, or disable limited downtime feature (can be done temporarily).

To disable the limited downtime feature for a single deployment you can use a switch:

```
ansible-playbook -e avoid_whole_environment_restart=False ...
```

CHAPTER 7

Configuration reference

```
# your project name, will create a namespace in docker with this name
# should contain only a-z, A-Z, 0-9 and _
COMPOSE_PROJECT_NAME=your_project_there

# git deployment specification for applications
GIT_SERVER=github.com
GIT_PROTO=https
GIT_USER=my-deploy-user
GIT_PASSWORD=
GIT_ORG_NAME=my-org-at-git-server

# User login and a group, who owns this repository files and all app files
# Keep empty to chose automatically.
#   Automatic chose priority:
#     - current sudo session user
#     - whoami if not in sudo session
#
APP_USER=
APP_GROUP_ID=

# login:group or id:gid of user who runs by default inside of a container
# All WRITABLE_DIRS will have this user as owner, so the webserver could allow to␣
↪upload files for example
#
# @override: This value can be overridden by repository configuration
DEFAULT_CONTAINER_USER=82:82

# use for testing to put each of your domain as a subdomain of eg. ".localhost"
DOMAIN_SUFFIX=.localhost
MAIN_DOMAIN=test
#ENFORCE_DEBUG_ENVIRONMENT=1
LETSENCRYPT_EMAIL=somemail@gmail.com

# health checking. Set a code to hide health checks endpoint behind a code (url␣
↪suffix eg. https://health.domain.org/some-test)
```

(continues on next page)

(continued from previous page)

```
HEALTH_CHECK_CODE=some-test

# dashboard
DASHBOARD_TITLE=Dashboard
APP_ADMIN_TOKEN=some-secret-huh

# health checks and dashboard
# if you use the uptimeboard, you may want to put providers there
MONITORING_PROVIDERS=UptimeRobot://some-token-there
# will be required to pass into path eg. http://health.your-app.org/some-code-here
HEALTH_CHECK_CODE=some-code-here

# SMTP gateway (passes mails through external gateway)
# your SMTP relay server address
SMARTHOST_ADDRESS=your-server.org
SMARTHOST_PORT=587
SMARTHOST_USER=login@your-server.org
SMARTHOST_PASSWORD=
# forward all e-mails, you can put here eg. allowed recipient domains
SMARTHOST_ALIASES=*

# backups
BACKUPS_URL=https://api.backups.your-project.org
BACKUPS_TOKEN=your-file-repository-token
BACKUPS_PASSPHRASE=with-this-encryption-key-backups-will-be-encrypted
BACKUPS_ENCRYPTION_METHOD=aes-128-cbc    # possible values: aes-128-cbc, aes-256-cbc,␣
↪see Bahub documentation
BACKUPS_CONTAINER=backup

# backups collections, collection per container/backup point
BACKUPS_PORTAINER_COLLECTION_ID=11111-2222-3333-4444

# watchtower
WATCHTOWER_INTERVAL=10
WATCHTOWER_SLACK_HOOK=https://mattermost.anarchista.net/hooks/
↪hxs9ebij57r15k1i6hz1dp6s6e
WATCHTOWER_IDENTIFIER="CIA-ZSP Watchtower"
```

CHAPTER 8

Cookbook

8.1 Guide to a perfect Wordpress page deployment

Following guide will give you following advantages:

- Automatic updates of Wordpress CMS
- Automatic SSL
- Automatic backup of the files
- Site theme managed by GIT, versioned
- Automatic creation of the configuration, database
- Do not need to copy anything manually to production, deploys first working version from prepared backups

8.1.1 1. Database

You need a database that will create your user with proper password, create internally a database.

```
version: "2"

volumes:
    mysql-data:

services:
    #
    # MySQL server
    #
    #  In ./containers/templates/source/mysql put JINJA2 templates of SQL files, they␣
↪will be compiled to ./containers/templates/compiled/mysql and run on MySQL start
    #  you can create users, databases with this method on MySQL startup, while␣
↪keeping the passwords in the .env file
    #
    #  --innodb_file_per_table=1 is a secure setting to keep tables separated (in␣
↪case of disk failure easier to recover data)
    #
    #  All MySQL databases data is stored in "./data/mysql". For backup configuration␣
↪use "docker_volumes"
    #  (docker_hot_volumes not recommended) with path /var/lib/mysql
    #
    db:
        image: mariadb
        volumes:
            - "mysql-data:/var/lib/mysql"     # use a named volume
            #- "./data/mysql:/var/lib/mysql"  # use a bind-mount (possibly slower,␣
↪but the files would be in one folder)
            - "./containers/templates/compiled/mysql:/docker-entrypoint-initdb.d/"
        command: mysqld --innodb_file_per_table=1
        environment:
            - MYSQL_USER=${MYSQL_USER}
            - MYSQL_PASSWORD=${MYSQL_PASSWORD}
            - MYSQL_DATABASE=${MYSQL_DATABASE}
            - MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD}
        labels:
            com.centurylinklabs.watchtower.enable: true


    #
    # MySQL admin panel
    #
    db_mysql_admin:
        image: phpmyadmin/phpmyadmin
        environment:
            - PMA_HOST=db

            # gateway configuration
            - VIRTUAL_HOST=dba.${MAIN_DOMAIN}${DOMAIN_SUFFIX}
            - VIRTUAL_PORT=80
            - LETSENCRYPT_HOST=dba.${MAIN_DOMAIN}${DOMAIN_SUFFIX}
            - LETSENCRYPT_EMAIL=${LETSENCRYPT_EMAIL}
        expose:
            - "80"
        labels:
            com.centurylinklabs.watchtower.enable: true
```

8.1.2 2. Backups configuration (optional)

See: Backups section of *Features*

Notice: It's optional. But you will need to copy your post images (uploads) to production manually, and later do manual backups of it. Consider having a $3 dollar/month 100GB VPS that stores backups of your files.

8.1.3 3. Adding theme to GIT

Create a git repository on the git hosting, eg. on bitbucket, github. Go into the eg. wp-content/themes/THEME-NAME, and make a git repository from it.

```
cd wp-content/themes/THEME-NAME
git init
git add .
git commit -m "Added to git"
git remote add origin REPOSITORY-URL-HERE
git push origin master
```

Now your Wordpress theme is in a git repository! You can use it, to version it, deploy updates to the server!

8.1.4 4. Connect theme git repository to environment

To automatically do a git pull on deployment of the environment, we can make the repository to be managed by environment.

In **./apps/repos-enabled/SOME-APP-NAME.sh** put a simple configuration script:

```
#!/bin/bash
# replace this with project name on github, bitbucket, gogs or other
export GIT_PROJECT_NAME=app.transprzyjazn

# the directory in ./apps/www-data/ to mount later in the YAML file to the container
export GIT_PROJECT_DIR=app.transprzyjazn
export WRITABLE_DIRS=""
```

Now you can fetch a new version with:

```
make update APP_NAME=SOME-APP-NAME
```

Or you can update all your themes for all websites:

```
make update_all
```

8.1.5 5. Make sure the environment is configured to work with GIT

In the **.env** file there is a section to configure link to a GIT server.

```
# git deployment specification for applications
GIT_SERVER=github.com
GIT_PROTO=https
GIT_USER=my-deploy-user
GIT_PASSWORD=
GIT_ORG_NAME=my-org-at-git-server
```

This will make all connected repositories in **./apps/repos-enabled/** to use this global GIT server by default. But each repository can override those settings in the shell configuration file.

8.1.6 6. Create YAML files and update .env file

When your website theme is in GIT, and is managed by environment you need to pass it as a volume to the Wordpress container. The path to the theme files would be **./apps/www-data/{{ PUT GIT_PROJECT_DIR VALUE HERE, IT CANNOT BE A VARIABLE }}**

Protip: Extract database password into .env for security reasons

Protip: Use MAIN_DOMAIN and DOMAIN_SUFFIX convention to have benefits of it

```
version: "2"
services:
    app_transprzyjazn:
        image: wolnosciowiec/wp-auto-update
        expose: ['80']
        volumes:
            - ./data/wordpress/app.transprzyjazn:/var/www/html
            - ./apps/www-data/app.transprzyjazn:/var/www/html/wp-content/themes/freddo
        environment:
            - WORDPRESS_DB_HOST=db
            - WORDPRESS_DB_USER=transprzyjazn
            - WORDPRESS_DB_PASSWORD=${DB_PASSWD_TRANSPRZYJAZN}
            - WORDPRESS_DB_NAME=transprzyjazn

            # gateway configuration
            - VIRTUAL_HOST=transprzyjazn.pl${DOMAIN_SUFFIX}
            - VIRTUAL_PORT=80
            - LETSENCRYPT_HOST=transprzyjazn.pl${DOMAIN_SUFFIX}
            - LETSENCRYPT_EMAIL=${LETSENCRYPT_EMAIL}
```

8.1.7 7. Have fun!

8.2 Guide to failure-safe server

The environment template natively chooses RiotKit's File Repository as the backups automation. Requirement: You need to have a server running File Repository, check file-repository.

Example *infrastructure.backup.yaml*

```
version: "2"
services:
    backup:
        image: wolnosciowiec/file-repository:bahub
        volumes:
            - "./containers/backup/cron:/cron:ro"
            - "./containers/backup/bahub.conf.yaml:/bahub.conf.yaml:ro"
            - "/var/run/docker.sock:/var/run/docker.sock"
        env_file:
            - ./.env

        environment:
            # On localhost will turn off automatic backups, so the dev will not␣
↪override production backups
```

(continues on next page)

(continued from previous page)

```
            # Still you can send backups to production backup server, so later you␣
↪can easily migrate applications
            # from developer environment to production environment.
            - DISABLE_SCHEDULED_JOBS=${IS_DEBUG_ENVIRONMENT}

        restart: on-failure
        mem_limit: 80000000 # 80M
        labels:
            com.centurylinklabs.watchtower.enable: true
```

Configuration

1. In **./containers/backup/bahub.conf.yaml** define backup definitions, a recovery plan, passwords. Take a look at the File Repository's documentation there: file-repository.docs.riotkit.org
2. All collection ids, passwords extract to **.env** file, example: **collection_id: "${BACKUPS_PORTAINER_COLLECTION_ID}"**
3. Schedule some backup jobs in **./containers/backup/cron** eg. **0 1 * * MON bahub backup db**

```
BACKUPS_URL=https://api.backups.your-project.org
BACKUPS_TOKEN=your-file-repository-token
BACKUPS_PASSPHRASE=with-this-encryption-key-backups-will-be-encrypted
# possible values: aes-128-cbc, aes-256-cbc, see Bahub documentation
BACKUPS_ENCRYPTION_METHOD=aes-128-cbc
# container name from the YAML file
BACKUPS_CONTAINER=backup
```

Recovery from backup

There are multiple cases when you need to recover multiple containers, or all containers from latest version from backup.

Example cases:

- Server failure, need to recreate the server
- Server was compromised "hacked", need to restore latest copy of data
- Migrating from development environment into production-ready, working live server
- Migrating from server to server

```
# will restore all services defined in bahub.conf.yaml into latest copy from backup␣
↪server
make recover_from_backup

./bin/backup
```

Note: Ansible deployment could attempt to take latest versions from backup when doing a first deploy on a server

8.3 Guide to deploying databases

Databases are not stateless, it's deploy is difficult than a deploy of a regular service. In this guide we focus on MySQL, but it's really up to you what database you will use.

Goals:

- Running up-to-date database engine

- Automatically made backup
- Migration of the data on deploy
- Administration panel for overview

8.3.1 1. Basic configuration

infrastructure.db.yml.example

```
version: "2"

volumes:
    mysql-data:

services:
    #
    # MySQL server
    #
    #  In ./containers/templates/source/mysql put JINJA2 templates of SQL files, they␣
↪will be compiled to ./containers/templates/compiled/mysql and run on MySQL start
    #  you can create users, databases with this method on MySQL startup, while␣
↪keeping the passwords in the .env file
    #
    #  --innodb_file_per_table=1 is a secure setting to keep tables separated (in␣
↪case of disk failure easier to recover data)
    #
    #  All MySQL databases data is stored in "./data/mysql". For backup configuration␣
↪use "docker_volumes"
    #  (docker_hot_volumes not recommended) with path /var/lib/mysql
    #
    db:
        image: mariadb
        volumes:
            - "mysql-data:/var/lib/mysql"     # use a named volume
            #- "./data/mysql:/var/lib/mysql"  # use a bind-mount (possibly slower,␣
↪but the files would be in one folder)
            - "./containers/templates/compiled/mysql:/docker-entrypoint-initdb.d/"
        command: mysqld --innodb_file_per_table=1
        environment:
            - MYSQL_USER=${MYSQL_USER}
            - MYSQL_PASSWORD=${MYSQL_PASSWORD}
            - MYSQL_DATABASE=${MYSQL_DATABASE}
            - MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD}
        labels:
            com.centurylinklabs.watchtower.enable: true


    #
    # MySQL admin panel
    #
    db_mysql_admin:
        image: phpmyadmin/phpmyadmin
        environment:
            - PMA_HOST=db

            # gateway configuration
```

(continues on next page)

(continued from previous page)

```
            - VIRTUAL_HOST=dba.${MAIN_DOMAIN}${DOMAIN_SUFFIX}
            - VIRTUAL_PORT=80
            - LETSENCRYPT_HOST=dba.${MAIN_DOMAIN}${DOMAIN_SUFFIX}
            - LETSENCRYPT_EMAIL=${LETSENCRYPT_EMAIL}
        expose:
            - "80"
        labels:
            com.centurylinklabs.watchtower.enable: true
```

8.3.2 2. Keep the database server up-to-date (optional)

You may want to enable the Watchtower to update the database image automatically, to do this please tag the database image with a label **com.centurylinklabs.watchtower.enable: true**

infrastructure.updates.yml.example

```
version: "2"
services:
    #
    # Automatically does a docker pull for tagged services with "com.centurylinklabs.
↪watchtower.enable" tag
    #
    autoupdater:
        image: v2tec/watchtower
        command: "--label-enable --interval ${WATCHTOWER_INTERVAL} --cleanup"
        volumes:
            - /var/run/docker.sock:/var/run/docker.sock
        environment:
            - WATCHTOWER_NOTIFICATIONS=slack
            - WATCHTOWER_NOTIFICATION_SLACK_HOOK_URL=${WATCHTOWER_SLACK_HOOK}
            - WATCHTOWER_NOTIFICATION_SLACK_IDENTIFIER=${WATCHTOWER_IDENTIFIER}
        labels:
            com.centurylinklabs.watchtower.enable: true
```

8.3.3 3. Have a backup (optional)

It's recommended to use File Repository as backup storage, but it's totally optional as it requires an additional server in your infrastructure.

Read more here: file-repository

Example bahub configuration:

You need to define **BACKUPS_DB_COLLECTION** in the **.env** file.

8.3.4 4. Automate migrations (optional)

Adding a new application to the network requires a manual user creation, database creation - this also can be automated optionally.

1. Create a _migrations database in the SQL
2. Enable **infrastructure.db_migrations.yml** configuration file

3. To execute SQL statements just right after deployment, put them into **./containers/migrations/prod** and enable the **db_migrations** configuration file
4. (Optionally) Use templating mechanism, put a SQL template in JINJA2 format in **./containers/templates/source**, so it will appear compiled in **/templates** inside of the container

infrastructure.db_migrations.yml.example

```
version: "2"
services:

    #
    # Executes migrations right after database starts, then container stops as it is no longer needed
    #
    db_updater:
        image: mkbucc/shmig:latest
        depends_on:
            - db
        volumes:
            - ./containers/migrations/prod:/sql:ro
            - ./containers/templates/compiled:/templates:ro
        command: -t mysql up
        environment:
            - PASSWORD=${MYSQL_ROOT_PASSWORD}
            - HOST=db_mysql
            - LOGIN=root
            - PORT=3306
            - DATABASE=_migrations
```

Example of automatically generated SQL files with using variables from .env file

./containers/templates/source/access.sql.j2

```
CREATE DATABASE IF NOT EXISTS zsp;
CREATE USER IF NOT EXISTS 'some_page'@'%' IDENTIFIED BY '{{ DB_PASSWD_SOME_PAGE }}';
GRANT ALL ON `some_page`.* TO 'some_page'@'%' IDENTIFIED BY '{{ DB_PASSWD_SOME_PAGE }}';
```

./containers/migrations/prod/1553701697-some-page.sql

```
-- Migration: some-page
-- Created at: 2019-03-27 16:46
-- ====  UP  ====

source /templates/access.sql;

-- ==== DOWN ====
```

8.3.5 5. When database will go down, show maintenance page (optional)

Infracheck is doing health checks of the infrastructure. The "infrastructure.health.yml" needs to be enabled at first.

Example health check

Please notice the on_each_down and on_each_up sections.

```
{
    "type": "port-open",
    "input": {
        "po_port": "3306",
        "po_host": "db_mysql",
        "po_timeout": "1"
    },
    "hooks": {
        "on_each_down": [
            "touch /maintenance/on"
        ],
        "on_each_up": [
            "rm -f /maintenance/on"
        ]
    }
}
```

8.4 Apps hosted on GIT and mounted as volumes

Applications based on CMS such as Wordpress and Drupal are not easy to pack into a docker container, as the user is allowed to manage them with web panel.

The solution for this situation could be to:

- Keep the constant files versioned in GIT (eg. website theme)
- Put all the files into the external backup, see *Guide to failure-safe server*

8.4.1 1. Configuring environment globally

The general concept is to have a single organization on github, bitbucket, gogs or other git server, and multiple repositories in it. But it is of course possible to have multiple and leave some of the global fields empty.

Just play with those environment variables in the .env file:

```
# git deployment specification for applications
GIT_SERVER=github.com
GIT_PROTO=https
GIT_USER=my-deploy-user
GIT_PASSWORD=
GIT_ORG_NAME=my-org-at-git-server
```

Remember, that all of those variables can be later overwritten by single repository configuration.

8.4.2 2. Making application manageable by environment

In *./apps/repos-enabled* create a bash script, it's actually not a script, but a configuration in shell syntax. Variables are inherited from **.env** file, so you can overwrite them there per-repository.

Example

```
#!/bin/bash
export GIT_PROJECT_NAME=some-app
export GIT_PROJECT_DIR=some-app
```

(continues on next page)

(continued from previous page)

```
export WRITABLE_DIRS="files store cache"

#
# Install the assets
#
post_update () {
    echo "I'm a post_update hook, i'm in $1 directory"
}
```

Good practice tip: It is good to have a read-only deployment user, to not leak your passwords with write-access

8.4.3 3. Deploying

Volume-based applications are deployed during *make deployment_pre*, when deploying with Ansible. To trigger a manual deploy, please check:

```
# deploy all git-volume-based applications
make update_all

# deploy single application
make update APP_NAME=some-app
```

What is the update command doing?

The update command is cloning the repository IF IT DOES NOT EXIST, in other case it is doing a git pull.

CHAPTER 9

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