GHENT
UNIVERSITY

FACULTY OF ENGINEERING
AND ARCHITECTURE

Installation Manual

Design Project - Runamic Ghent

Group 16
Depauw Hendrik
van Herwaarden Lorenz
Aelterman Nick
Cammaert Olivier
Deweirdt Maxim
Dox Gerwin
Neuville Simon
Uyttersprot Stiaan

Contents

1 Introduction **1**

2 Docker **1**

3 Nginx + Letsencrypt **2**

4 SonarQube + MySQL **2**

5 Jenkins **3**

6 MongoDB **3**

7 Optional tools **4**

8 DRIG_client Configuration **5**
- 8.1 Jenkins 5
- 8.2 SonarQube 5

9 DRIG_server Configuration **5**
- 9.1 Jenkins 5
- 9.2 SonarQube 6

1 Introduction

This document provides instructions on how to set up the backend and all necessary services for Runamic Ghent. This manual assumes the system administrator uses a server (both a dedicated server or VPS are fine) with a static IP address and Ubuntu 16.04 installed. Every service runs in its own Docker container. This means it should be possible to deploy the backend to other Linux/Unix based systems and even Windows based hosts as long as an up-to-date Docker version is available. However, this has not been tested and problems may occur if using an OS other than Ubuntu 16.04.

As mentioned, Docker is an important and necessary part of the setup. It makes it very easy to reset installations, spin up new instances when something goes wrong, migrate and even scale out. Docker also isolates services from each other and the host, thus providing an extra layer of security. This setup provides a lot of the advantages of virtualization with less overhead.

The following services need to be deployed:

- Nginx: used as a reverse proxy to access different services with a domain name and subfolder instead of an IP address and ports.
- SonarQube: continuous code quality tool.
- MySQL: database required to run SonarQube.
- Jenkins: continuous integration and continuous delivery tool.
- MongoDB: NoSQL database.
- [OPTIONAL] Portainer: a simple management tool for Docker accessible via the browser.
- [OPTIONAL] netdata: real-time performance and health monitoring tool.

2 Docker

Before doing anything else, it is recommended to update the package database. Next, the GPG key for the official Docker repository has to be added to the system, followed by the Docker repository. Now update the package database once again.
Finally install Docker and check if it is running.

```
sudo apt-get update
sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --
   recv-keys 58118E89F3A912897C070ADBF76221572C52609D
sudo apt-add-repository 'deb https://apt.dockerproject.org/repo
   ubuntu-xenial main'
sudo apt-get update

sudo apt-get install -y docker-engine
sudo systemctl status docker
```

3 Nginx + Letsencrypt

This section helps setting up an Nginx reverse proxy with a built-in letsencrypt client that automates free SSL server certificate generation and renewal processes. It also contains fail2ban for intrusion prevention.

The following command suffices to get the container up and running. Only the values between tags should be replaced with actual parameters. The container is available in the Docker Repository as linuxserver/letsencrypt.

```
docker run -d \
  --privileged \
  --name=letsencrypt \
  -v <path to data>:/config \
  -e PGID=<gid> -e PUID=<uid>  \
  -e EMAIL=<email> \
  -e URL=<url> \
  -e SUBDOMAINS=<subdomains> \
  -p 443:443 \
  -p 80:80
  linuxserver/letsencrypt
```

Defining routes and options for the reverse proxy can be done by adjusting the configuration file found in /config/nginx/site-configs/default. For reference, the Runamic Ghent configuration has been included in the *default* file. SSL certificates are generated and renewed automatically and allow for secure communication between the server and clients.

https://hub.docker.com/r/linuxserver/letsencrypt/

4 SonarQube + MySQL

SonarQube is a tool for continuous code quality. It provides a webinterface where developers can review test results, code coverage and other metrics concerning their code. By default, the docker image will use an embeded database upon creation. However, it is strongly discouraged to run in a production environment and a MySQL database is the recommended alternative. Since two containers need to be deployed and linked, a docker compose file (included as *docker-compose.yaml*) has been provided that takes care of this:

```
sonarqube:
  image: sonarqube
  ports:
   - "9000:9000"
   - "3306:3306"
  environment:
   - SONARQUBE_JDBC_USERNAME=sonar
   - SONARQUBE_JDBC_PASSWORD=sonar
   - SONARQUBE_JDBC_URL=jdbc:mysql://localhost:3306/sonar?useUnicode
       =true&characterEncoding=utf8&rewriteBatchedStatements=true

db:
  image: mysql
```

```
  net: container:sonarqube
  environment:
   - MYSQL_ROOT_PASSWORD=sonar
   - MYSQL_DATABASE=sonar
   - MYSQL_USER=sonar
   - MYSQL_PASSWORD=sonar
```

Database usernames and passwords should be adjusted for security concerns.

This file can be used to ship the two containers. When up and running a few settings still need to be adjusted in order to use a reverse proxy.

```
docker-compose up -d
sudo vi <path to data>/sonar/conf/sonar.properties
sonar.web.host=0.0.0.0
sonar.web.port=9000  (or another port if you want)
sonar.web.context='/sonar'
```

Both containers are provided in the standard Docker Repository as library/mysql and library/sonarqube

https://hub.docker.com/_/sonarqube/
https://hub.docker.com/_/mysql/

5 Jenkins

Since there were a lot of problems during Android build jobs and different SDK versions with the official Jenkins template, there has been opted for a custom Dockerfile for the Jenkins container. It can be found as attached file *Dockerfile_jenkins* and following commands are necessary to build and ship the container:

```
docker build -t groep16/jenkinsci -f <path to Dockerfile>/
   Dockerfile_jenkins .
sudo docker run -d -p 8089:8080 -p 50001:50000 \
    -v <path to data>:/var/jenkins_home \
    -v /usr/bin/docker:/usr/bin/docker \
    -v /var/run/docker.sock:/var/run/docker.sock  \
    -e JENKINS_OPTS='--prefix=/jenkins' \
    --name=jenkinsci groep16/jenkinsci

docker exec -u root jenkins /bin/chmod -v a+s \$(which docker)
```

This makes the application accessible on */jenkins* and also gives the container access to the docker commando from the host. This is needed for building and deploying the Django server as a container.

6 MongoDB

The MongoDB container is ideal for the storage of JSON data and provides a NoSQL database where route ratings are stored. The official container library/mongo from the Docker Repository is used.

The necessary steps to deploy are:

```
docker run --name mongodb -p 27017:27017 -d mongo --auth
docker exec -it mongodb mongo admin
>db.createUser({
    user: 'lopeningent',
    pwd: 'xxxxxxx',
    roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
});
>use edges
>db.createUser({
    user: 'server',
    pwd: 'xxxxxx',
    roles: [ { role: "readWrite", db: "edges" } ]
});
>use DRIG_userdata
>db.createUser({
    user: 'client',
    pwd: 'xxxxxx',
    roles : [ { role: "readWrite", db: "DRIG_userdata" } ]
});
>use DRIG_unittest
>db.createUser({
    user: 'client',
    pwd: 'xxxxxx',
    roles : [ { role: "readWrite", db: "DRIG_unittest" } ]
});
```

https://hub.docker.com/_/mongo/

7 Optional tools

Two optional tools were used to manage and monitor the server and Docker and to easily give developers console access to containers: Portainer and netdata.
Both are very easy to deploy and available in the Docker Repository under titpetric/netdata and portainer/portainer. Following commando's are needed to get the containers up and running:

```
docker run -d --cap-add SYS_PTRACE \
           -v /proc:/host/proc:ro \
           -v /sys:/host/sys:ro \
           -p 19999:19999 titpetric/netdata

docker run -d -p 9001:9000 portainer/portainer
```

https://hub.docker.com/r/titpetric/netdata/
https://portainer.readthedocs.io/en/latest/deployment.html

8 DRIG_client Configuration

8.1 Jenkins

Creating a build job in Jenkins for the Android application is a quick and easy process. Just set up a new Freestyle project, refer to the git repository, provide the branch and optionally activate the GitHub hook trigger for automated builds on a push. Choose to use Gradle Wrapper in the Build section and provide the path to the gradle file (DynamicrunninginGhent/build.gradle). The following tasks are necessary: clean, createDebugCoverageReport, jacocoTestReport, sonarqube.

8.2 SonarQube

The SonarQube configuration is handled in the project's build.gradle file. The only parameters that should be changed when migrating are *sonar.host.url, sonar.login, sonar.password.*

9 DRIG_server Configuration

9.1 Jenkins

Creating a build Job for the Django server is a bit more complicated. Again, start of with a Freestyle project, refer to the git repository and provide the branch.

Each Django server instance is packaged in a Docker container. This allows to easily run multiple versions concurrently which is very handy for the development team. Add a shell script in the Build section and paste the following code:

```
if [ "\$(docker ps -q -f name=djangoserver)" ]; then
        docker stop djangoserver
    docker rm djangoserver
fi
chmod u+x docker-entrypoint.sh
docker build -t groep16/djangoserver .
docker run -d -p 8001:8000 --name=djangoserver groep16/djangoserver
sleep 960s
docker cp djangoserver:/srv/djangoserver/nosetests.xml /var/
   jenkins_home/workspace/DRIG_server\ -\ develop/djangoserver
docker cp djangoserver:/srv/djangoserver/pylint.txt /var/
   jenkins_home/workspace/DRIG_server\ -\ develop/djangoserver
docker cp djangoserver:/srv/djangoserver/coverage.xml /var/
   jenkins_home/workspace/DRIG_server\ -\ develop/djangoserver
```

This checks for old containers and removes them before continuing. Next a docker container is build and the Django application is started. The *sleep* commando is necessary if we want to export code coverage and test results after the build to SonarQube. Since it takes some time to generate these reports, Jenkins needs to wait before continuing.

If all tests are sucessful, the container should be up and running and ready for requests.

9.2 SonarQube

Integration with SonarQube is accomplished through the SonarQube Scanner plugin in Jenkins. Add the following lines to the Analysis properties in the Build section and adjust were needed:

```
sonar.projectKey=drig:develop
sonar.projectName=DRIG_server - develop
sonar.projectVersion=1.0

sonar.language=py
sonar.projectBaseDir=djangoserver
sonar.sources=.
sonar.exclusions=**/migrations/**,**/test/**,integration/**,fabfile.
    py,scent.py,**/scripts/**,**/preprocess/**

sonar.sourceEncoding=UTF-8
sonar.python.pylint.reportPath=djangoserver/pylint.txt
sonar.python.xunit.reportPath=djangoserver/nosetests.xml
sonar.python.coverage.reportPath=djangoserver/coverage.xml
sonar.core.codeCoveragePlugin=cobertura
sonar.login=jenkins
sonar.password=jenkins
```