HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Rapport de laboratoire: http infrastructure

Département : Technologies de l'Information et de la Communication (TIC)
Unité d'enseignement : Réseau(RES)

Auteurs : Fabrice Mbassi
Professeur : O.Liechti
Assistant : A.Adrien & D.Palumbo
Classe : RES C
Date : 2 juin 2020

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts
Western Switzerland

Table des matières

1 serveur HTTP statique avec apache httpd

1.1 Labo HTTP (1) : Serveur apache httpd "dockerisé" servant du contenu statique

Pour cette partie nous avons suivie les étapes suivantes :

- https ://github.com/fabano237/RES_HTTP_INFRA_2020.
- Créer notre Dockerfile contenant notre recette pour la construction de notre image aphache php. Notre fichier dans un premier temps comporte les commandes suivantes :

```
FROM php:7.2-apache
COPY src/ /var/www/html/
```

- Nous avons build l'image puis lancer notre contener avec la commande : **docker run -d -p 9090 :80 php :7.2-apache**.
- Vu que nous ne sommes pas dans une machine virtuelle nous pouvons directement tester avec la commande : **telnet 172.17.0.2 80** c'est à dire sans le port mapping 9090.

```
fabano@fabano-WRT-WX9:~/Documents/sm2/res/RES_HTTP_INFRA_2020/docker-images/apache-php-image$ telnet 172.17.0.2 80
Trying 172.17.0.2...
Connected to 172.17.0.2.
Escape character is '^]'.
GET / HTTP/1.0
HTTP/1.1 408 Request Timeout
Date: Thu, 14 May 2020 12:33:22 GMT
Server: Apache/2.4.38 (Debian)
Content-Length: 297
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>408 Request Timeout</title>
</head><body>
<h1>Request Timeout</h1>
<p>Server timeout waiting for the HTTP request from the client.</p>
<hr>
<address>Apache/2.4.38 (Debian) Server at 172.17.0.2 Port 80</address>
</body></html>
Connection closed by foreign host.
```

- Alors pour mieux visualiser le fonctionnement de notre serveur, nous avons édité le fichier index.html dans le conteneur avec les commandes suivante :
  - **docker exec -it kind_tharp /bin/bash** pour lancer le bash dans le conteneur.
  - **echo "<h2>Coucou<h2>" > index.html**. Pour mettre un peu de contenu texte dans la page web. Il suffit alors d'aller sur un navigateur et taper : **172.17.0.2 :80**. et le texte est affiché.

2 serveur HTTP dynamique avec express.js

2.1 Labo HTTP (2a) : Noeud d'application "dockerisée"

Dans cette partie nous construisons un Noeud application web dynamique. pour cette partie il est essentielle de faire une recette de cuisine, notre Dockerfile avec le contenu suivant :

```
FROM node:8.10.0 # version de node utilisée

COPY src /opt/app # copy du dossier src dans /opt/app de
↪  l'image

 # commande lancer lors de l'execution du contenuer
CMD ["node", "/opt/app/index.js"]
```

Il fudra donc creer un sous repertoire src/ dans lequel sera contenue les fichiers :

- package.json générer par la commande npm init (utilitaire lors du démarage d'une nouvelle application node js) le contenu de ce fichier est :

```
{
  "name": "students",
  "version": "0.1.0",
  "description": "fabano: first instrumentation",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Fabrice Mbassi",
  "license": "ISC",
  "dependencies": {
    "chance": "^1.1.5"
  }
}
```

- A noter que la dépendance du module chance a été créée via la commande : **npm install –save chance**.
- Puis ajouter le script index.js donc on va lui passer le contenu suivant :

```
// utilisation du module
var Chance = require('chance');

// contructeur via le new module
var chance = new Chance();

// affichage sur console des noms aléatoire
console.log("Bonjour " + chance.name());
```

Nous pouvons donc exécuter notre application node via la commande : **node index.js**. Pour la suite on peu ainsi créer notre image et lancer un conteneur avec les commandes suivantes :

- **docker build -t res/express_students .**
- **docker run res/express_students**

Nous pouvons observer le résultat suivant :

```
fabano@fabano-WRT-WX9:~/Documents/sm2/res/RES_HTTP_INFRA_2020/docker-images/express-image/src$ docker run res/express_students
BonjourCole Hayes
fabano@fabano-WRT-WX9:~/Documents/sm2/res/RES_HTTP_INFRA_2020/docker-images/express-image/src$ docker run res/express_students
BonjourLogan Casey
fabano@fabano-WRT-WX9:~/Documents/sm2/res/RES_HTTP_INFRA_2020/docker-images/express-image/src$ docker run res/express_students
BonjourRoy Fox
fabano@fabano-WRT-WX9:~/Documents/sm2/res/RES_HTTP_INFRA_2020/docker-images/express-image/src$
```

Finalement nous pouvons contrôler à l'intérieur d'un conteneur si tout s'est passé

comme prévu avec la commande :
docker run -it res/express_students /bin/bash

```
fabano@fabano-WRT-WX9:~/Documents/sm2/res/RES_HTTP_INFRA_2020/docker-images/express-image$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
fabano@fabano-WRT-WX9:~/Documents/sm2/res/RES_HTTP_INFRA_2020/docker-images/express-image$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                      PORTS               NAMES
af3ef88b1041        res/express_students   "node /opt/app/index…"   2 minutes ago       Exited (0) 2 minutes ago                        dazzling_thar
p
5eaa14357dda        res/express_students   "node /opt/app/index…"   2 minutes ago       Exited (0) 2 minutes ago                        gallant_north
cutt
fabano@fabano-WRT-WX9:~/Documents/sm2/res/RES_HTTP_INFRA_2020/docker-images/express-image$ docker run -it res/express_students /bin/bash
root@53aa323e0ae4:/# node -v
v8.10.0
root@53aa323e0ae4:/# cd /opt/
app/         yarn-v1.5.1/
root@53aa323e0ae4:/# cd /opt/app/
root@53aa323e0ae4:/opt/app# ls
index.js  node_modules  package.json
root@53aa323e0ae4:/opt/app#
```

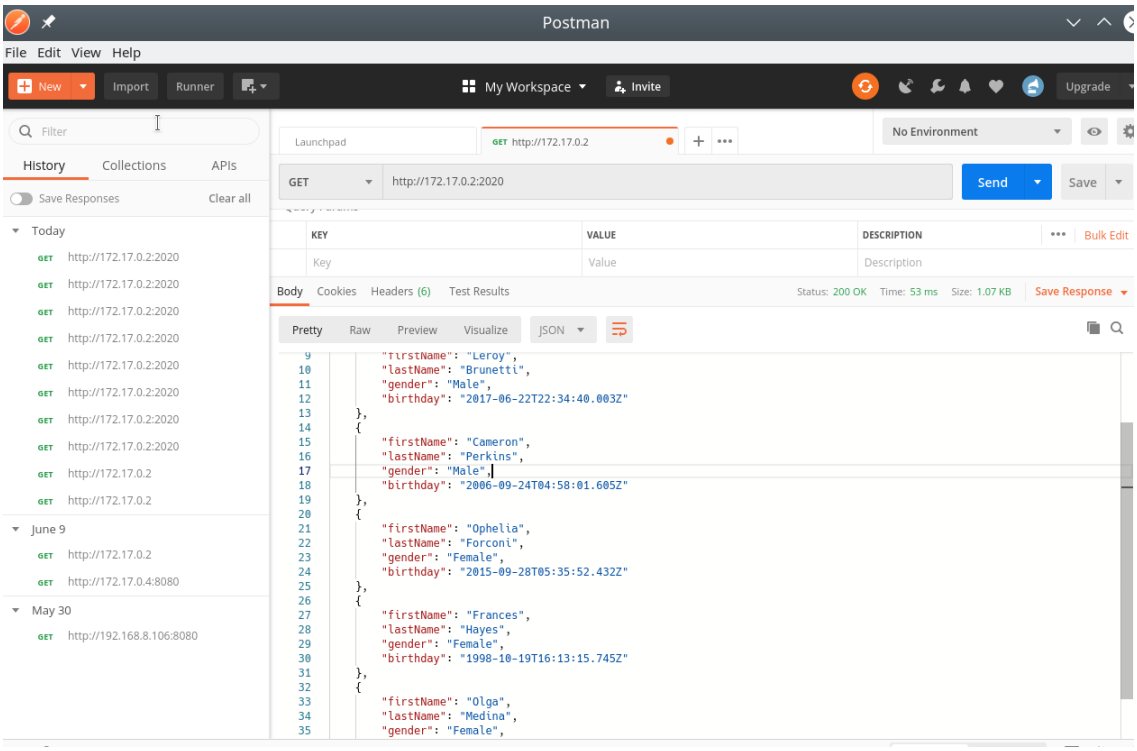
2.2 Labo HTTP (2b) : Application express "dockerisée"

Dans cette partie nous instrumentons les classes de base du module standard du serveur http. Nous manipulons les objets request et respons pour accéder à un serveur ou envoyer des données au client. Dans le cadre de notre laboratoire nous utilisons le framework express. Pour ce faire il faut installer express avec la commande : **nmp install –save express**. Le code utilisé pour notre démonstration est le suivant :

```
var Chance = require('chance');

var chance = new Chance();

const express = require('express');
const app = express();
const port = 2020;

app.get('/', (req, res) => res.send(generateStudents()));
app.listen(port,() => console.log(`Accepting HTTP request on:
 ↪  ${port}`));

function generateStudents(){
    var numberOfstudents = chance.integer({min:0, max: 10});
    console.log(numberOfstudents);
    var students = [];

    for(var i = 0; i < numberOfstudents; ++i){
        var gender = chance.gender();
        var birthYear = chance.year({min: 1993, max: 2020});

        students.push({firstName: chance.first({gender: gender}),
                       lastName:  chance.last(),
                       gender:    gender,
                       birthday:  chance.birthday({year:
                        ↪  birthYear})});
    };
```

```
    console.log(students);
    return students;
}
```

Lorsque l'on build l'image et on démarre notre conteneur on peut le tester de la même façon que les fois précédentes. Pour tester le bon fonctionnement nous avons décidé d'introduire l'application **POSTMAN**. voici le résultat obtenu :



3 proxy inverse avec apache (configuration statique)

Dans cette partie nous allons devoir configure notre serveur apache pour faire du reverse proxy affin de servir les serveurs développés dans la première et deuxième partie.
Dans ce chapitre nous allons lancer nos deux conteneurs précédents de telle sorte qu'ils soient accessible depuis des points d'entrées configuré dans notre reverse proxy.
Le fichier de configuration a le contenu suivant :

```
<VirtualHost *:80>
        ServerName demo.res.ch


        #ErrorLog ${APACHE_LOG_DIR}/error.log
        #CustomLog ${APACHE_LOG_DIR}/access.log combined


        proxyPass "/api/students/" "http://172.17.0.3:2020/"
        proxyPassReverse "/api/students/" "http://172.17.0.3:2020/"


        proxyPass "/" "http://172.17.0.2:80/"
        proxyPassReverse "/" "http://172.17.0.2:80/"

</VirtualHost>
```

A noter bien que les adresses sont codées en dure dû au faite que nous avons serveur avec une configuration statique.
Voici le Dockerfile pour le lancement de notre conteneur et la construction d'image.

```
FROM php:7.2-apache

#copy de la config dans la config du conteneur
COPY conf/ /etc/apache2

#Activation des modules apache
RUN a2enmod proxy proxy_http
#Activation des sites correspondant aux configs
RUN a2ensite 000-* 001-*
```

les différents conteneurs sont donc lancés de la manière suivante :

```
#docker run -d res/apache_php
#docker run -d res/express_students
#docker run    res/apache_rp
```

Il faudrait éditer aussi le fichier /etc/hosts de notre système de fichier en ajoutant la ligne 172.17.0.4 demo.res.ch pour effectuer une résolution de nom de notre serveur.
Le test est fait à partir d'un navigateur :
http ://demo.res.ch :80 affichage du contenu statique
http ://demo.res.ch :2020/api/students/ affichage contenu dynamique.

4 requêtes AJAX avec JQuery

Dans cette étape nous avons écrit une requête ajax avec la bibliothèque JQuery. La dite requête nous permet de faire interagir la partie backend et frontend. Pour cela nous devons pouvoir lier le contenu de notre serveur statique avec celui de notre serveur dynamique. nous changeons donc ainsi une partie de notre texte du contenu statique avec du contenu dynamique soit un nom et prénom de notre liste des noms générée dynamiquement. Pour lier les deux il est impératif d'utiliser l'attribut class ou id. Dans notre cas nous avons choisie d'utiliser l'attribut class.
Le contenu du script js(students.js) est le suivant :

```
$(function() {
    console.log("geting students Name");
    function getStudentsNames() {
        $.getJSON("/api/students/", function(students) {
            var message = "Nobody is here";
            console.log(students);
            if(students.length > 0){
                message = students[0].firstName + " " +
                ↪  students[0].lastName;
            }
            $(" .masthead-subheading").text(message);

        });
    };
    getStudentsNames();
    // contenue affiché toutes les 2s
    setInterval(getAStudentsNames, 2000);
});
```

Dans le fichier index.html nous intégré le sript js via la balise suivante :

```
<!-- Custom script to load students-->
        <script src="js/students.js"></script>
```

Le test est fait à partir d'un navigateur :
http ://demo.res.ch :80 affichage du contenu statique avec changement dynamique via la raquête ajax au niveau de class **"masthead-subheading"**

5 Configuration du proxy inverse dynamique

L'objectif de cette partie est de résoudre la problématique du reverse proxy statique qui nous amène à coder en dure les adresses de nos différents conteneur.
Pour cela nous avons fait scripte pour nous permettre de charger de façon dynamique nos adresse ip de conteneurs qui seront passées en paramètre à la commande d'exécution de notre conteneur du reverse proxy dynamique.

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

#Add setup RES lab
echo "Setup for the RES lab!!"
echo "static app URL: $STATIC_APP"
echo "dynamic app URL: $DYNAMIC_APP"

php /var/apache2/templates/config-template.php >
/etc/apache2/sites-available/001-apache_reverse_proxy.conf

#Apache get grumpy about PID files pre-existing
rm -f /var/run/apache2/apache2.pid

exec apache2ctl -DFOREGROUND
```

Il faut noter la présence du fichier config-template.php qui contient toutes les configurations qui seront chargée lors du démarrage du conteneur.

```
    <?php
    $static_app  = getenv('STATIC_APP');
    $dynamic_app = getenv('DYNAMIC_APP');
?>
<VirtualHost *:80>
        ServerName demo.res.ch


        proxyPass '/api/students/' 'http://<?php print
↪   "$dynamic_app"?>/'
        proxyPassReverse '/api/students/' 'http://<?php print
↪   "$dynamic_app"?>/'


        proxyPass '/' 'http://<?php print "$static_app"?>/'
        proxyPassReverse '/' 'http://<?php print "$static_app"?>/'

</VirtualHost>
```

le fichier de configuration du reverse proxy static a été remplacé par celui ci :

```
<VirtualHost *:80>
        ServerName demo.res.ch


        #ErrorLog ${APACHE_LOG_DIR}/error.log
        #CustomLog ${APACHE_LOG_DIR}/access.log combined


        proxyPass "/api/students/" "http://express_dynamic:2020/"
        proxyPassReverse "/api/students/"
        ↪  "http://express_dynamic:2020/"


        proxyPass "/" "http://apache_static:80/"
        proxyPassReverse "/" "http://apache_static:80/"

</VirtualHost>
```

cette configuration nous permet de lancer nos différent conteneur dans l'ordre suivant :

```
docker kill $(docker ps -q)
docker rm $(docker ps -a -q)

docker build -t res/apache_php apache-php-image/
docker build -t res/express_students express-image/
docker build -t res/apache_rp apache-reverse-proxy/

docker run -d res/apache_php
docker run -d res/apache_php
docker run -d res/apache_php
docker run -d --name apache_static res/apache_php

docker run -d res/express_students
docker run -d res/express_students
docker run -d --name express_dynamic res/express_students

docker inspect express_dynamic | grep -i ipaddr
docker inspect apache_static | grep -i ipaddr

docker run  -e STATIC_APP=172.17.0.5:80 -e DYNAMIC_APP=172.17.0.8:2020
↪  --name apache_rp res/apache_rp
```

Il faut pas oublié de réadapter notre fichier hosts pour le **DNS : 172.17.0.9 demo.res.ch**
Le test ce fait de la même façon que l'étape précédente soit **http ://demo.res.ch/** dans un navigateur.
Nous avons eu un problème de variable d'environnement non défini dans notre version d'apache : APACHE_RUN_DIR. Nous avons corrigé ce problème en passant à la commande de scripte apache2ctl au lieu de apche2 comme dans les webcast.

6 Étapes supplémentaires

6.1 Load balancing : multiple server nodes

Le load balancing permettra de répartir la charge lorsque nous avons plusieurs instances de docker d'un même site Web.
la majeur partie de la configuration se fait dans le fichier de config php voici son contenu :
source : https ://support.rackspace.com/how-to/simple-load-balancing-with-apache/

```
<?php
        $static_app1= getenv('STATIC_APP1');
        $static_app2 = getenv('STATIC_APP2');
        $static_app3 = getenv('STATIC_APP3');
        $dynamic_app1 = getenv('DYNAMIC_APP1');
        $dynamic_app2 = getenv('DYNAMIC_APP2');
        $dynamic_app3 = getenv('DYNAMIC_APP3');

?>

<VirtualHost *:80>
        ProxyRequests off

        ServerName demo.res.ch
        ServerAlias www.demo.res.ch

        <Location /balancer-manager>
                SetHandler balancer-manager
        </Location>

        ProxyPass /balancer-manager !

        <Proxy balancer://dynamic-cluster>
                # WebHead1
                BalancerMember 'http://<?php print
↪   "$dynamic_app1"?>'
                # WebHead2
                BalancerMember 'http://<?php print
↪   "$dynamic_app2"?>'
                # WebHead3
```

```
                BalancerMember 'http://<?php print
↪  "$dynamic_app3"?>'

                # Security "technically we aren't blocking
                # anyone but this is the place to make
                # those changes.
                Require all granted

                # Load Balancer Settings
                # We will be configuring a simple Round
                # Robin style load balancer.  This means
                # that all webheads take an equal share of
                # of the load.
                ProxySet lbmethod=byrequests
        </Proxy>
        # Point of Balance dynamic
        ProxyPass '/api/students/' 'balancer://dynamic-cluster/'
        ProxyPassReverse '/api/students/'
↪  'balancer://dynamic-cluster/'

        # We do the same for the static cluster
        <Proxy balancer://static-cluster>
                # WebHead1
                BalancerMember 'http://<?php print "$static_app1"?>'
                # WebHead2
                BalancerMember 'http://<?php print "$static_app2"?>'
                # WebHead3
                BalancerMember 'http://<?php print "$static_app3"?>'

                # Security "technically we aren't blocking
                # anyone but this is the place to make
                # those changes.
                Require all Granted

                # Load Balancer Settings
                ProxySet lbmethod=byrequests
        </Proxy>
        # Point of Balance static
        ProxyPass  '/' 'balancer://static-cluster/'
        ProxyPassReverse '/' 'balancer://static-cluster/'


</VirtualHost>
```

Nous avons donc créé un réseaux de 3 serveurs static et dynamic pour faire le test :

```
docker run -d --name apache-static1 res/apache_php
```

```
docker run -d --name apache-static2 res/apache_php
docker run -d --name apache-static3 res/apache_php

docker run -d --name express-dynamic1 res/express_students
docker run -d --name express-dynamic2 res/express_students
docker run -d --name express-dynamic3 res/express_students

docker run -d -e STATIC_APP1=172.17.0.2:80 -e
 ↪  STATIC_APP2=172.17.0.3:80 -e STATIC_APP3=172.17.0.4:80 \
-e DYNAMIC_APP1=172.17.0.5:2020 -e DYNAMIC_APP2=172.17.0.6:2020 -e
 ↪  DYNAMIC_APP3=172.17.0.7:2020 --name apache-reverse-proxy
 ↪  res/apache_rp
```

Pour verifier le fonctionnement il suffit d'aller sur : http ://demo.res.ch/balancer-manager

6.2 Docker Management UI

Pour cette étape nous voulons implémenter un moyen d'avoir une interface utilisateur pour la gestion de nos conteneur, image docker. Nous avons trouvé la documentation ici : https ://www.portainer.io/installation/. Avec les commande suivante nous chargeons l'image portainer et en suite nous lançons un conteneur qui va nous permettre d'avoir une interface facile à utiliser pour le monde "dockerisé"

```
docker volume create portainer_data

    docker run -d -p 9000:9000 --name portainer --restart always
     ↪  -v /var/run/docker.sock:/var/run/docker.sock -v
     ↪  portainer_data:/data portainer/portainer
```

Pour tester il suffit d'aller sur un navigateur et saisir le lien suivant : http ://localhost :9000

7 Signatures

Yverdon-les-Bains le 2 juin 2020

Fabrice Mbassi