

# **Rapport de laboratoire**

## **WEB Infrastructure Lab**

Amine Tayaa – Valentin Schaad – Sébastien Henneberger – (Benoît Zuckschwerdt)

1<sup>er</sup> juin 2015

**Table des matières :**

1	Introduction.....	3
2	Mise en place du front-end.....	4
3	Mise en place du back-end.....	7
4	Mise en place du serveur (reverse proxy + load-balancer) .....	9
5	Tests .....	11
6	Conclusion .....	12

## 1 Introduction

Le but de ce laboratoire est de créer une infrastructure web et d'implémenter notamment un reverse proxy et un répartiteur de charge (load-balancer).

En tant qu'utilisateur, on devra pouvoir ouvrir une page web dont le contenu statique est délivré par le container implémentant le front-end. Un script s'exécutera dans la page web affichée par le client qui effectuera des requêtes AJAX pour récupérer un contenu dynamique délivré par le back-end.

Le front-end doit implémenter les sticky-sessions, c'est-à-dire qu'une fois qu'un client s'est connecté à un front-end, il doit se souvenir de lui et garder la même session pour leur communication.

Autrement-dit, le container utilisé sera le même pour toute la session.

A l'inverse, le back-end coupera la connexion après toute communication. C'est-à-dire, que ce ne sera pas forcément le même container implémentant un back-end qui sera attribué pour le même utilisateur au cours du temps.

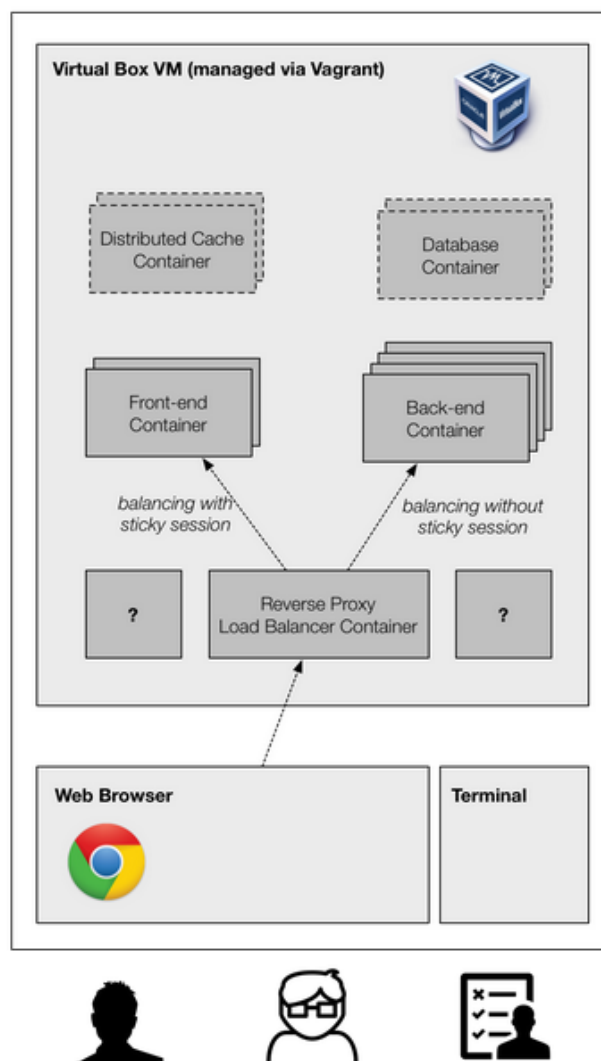


Schéma 1 Infrastructure Web

## 2 Mise en place du front-end

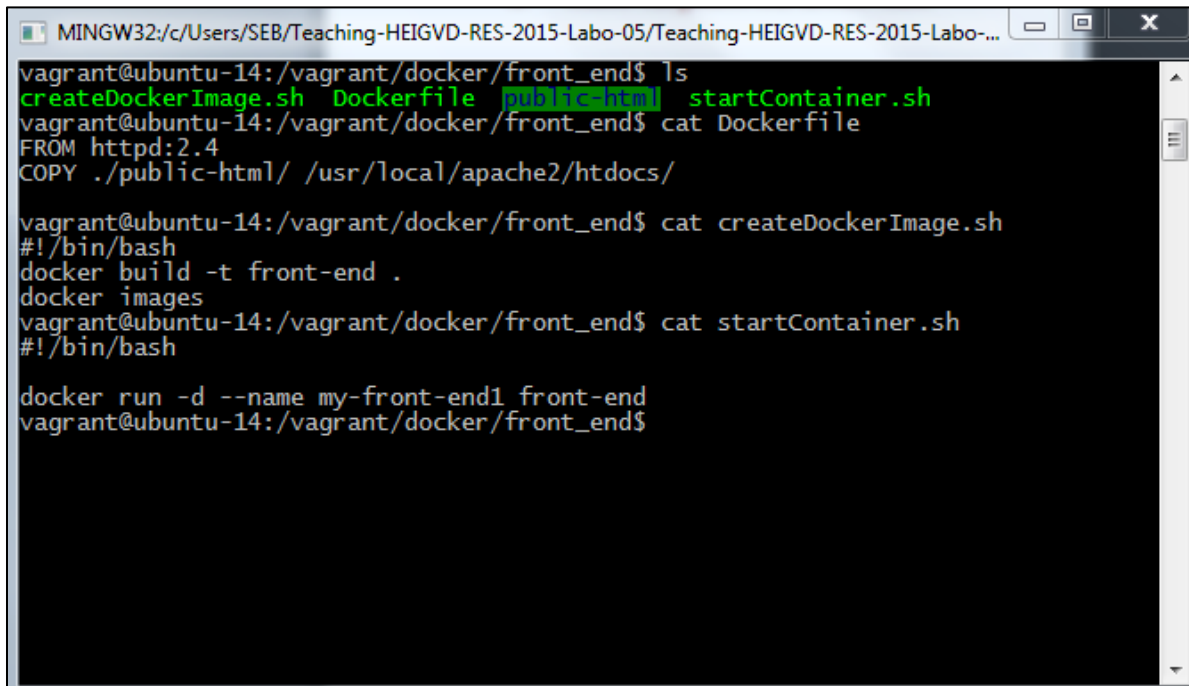
Nous avons utilisé l'image httpd disponible dont la documentation est disponible sous :

[https://registry.hub.docker.com/\\_/httpd/](https://registry.hub.docker.com/_/httpd/)

Il s'agit d'une image officielle d'un serveur Apache http.

Du coup, on se crée un fichier nommé Dockerfile qui sera utilisé pour la création de l'image, voir la capture ci-dessous.

Pour plus de simplicité, on s'est créé un script permettant de créer l'image docker et de démarrer un container avec cette image, voir la capture ci-dessous :



```
MINGW32/c/Users/SEB/Teaching-HEIGVD-RES-2015-Labo-05/Teaching-HEIGVD-RES-2015-Labo-...
vagrant@ubuntu-14:/vagrant/docker/front_end$ ls
createDockerImage.sh Dockerfile public-html startContainer.sh
vagrant@ubuntu-14:/vagrant/docker/front_end$ cat Dockerfile
FROM httpd:2.4
COPY ./public-html/ /usr/local/apache2/htdocs/

vagrant@ubuntu-14:/vagrant/docker/front_end$ cat createDockerImage.sh
#!/bin/bash
docker build -t front-end .
docker images
vagrant@ubuntu-14:/vagrant/docker/front_end$ cat startContainer.sh
#!/bin/bash

docker run -d --name my-front-end1 front-end
vagrant@ubuntu-14:/vagrant/docker/front_end$
```

Ensuite, il faut faire la page web avec un fichier index.html.

Il s'agit d'une simple page web, affichant un titre. Mais, elle possède un script javascript qui toutes les secondes interroge le back-end pour obtenir un contenu dynamique (représentation JSON de /api/message) :

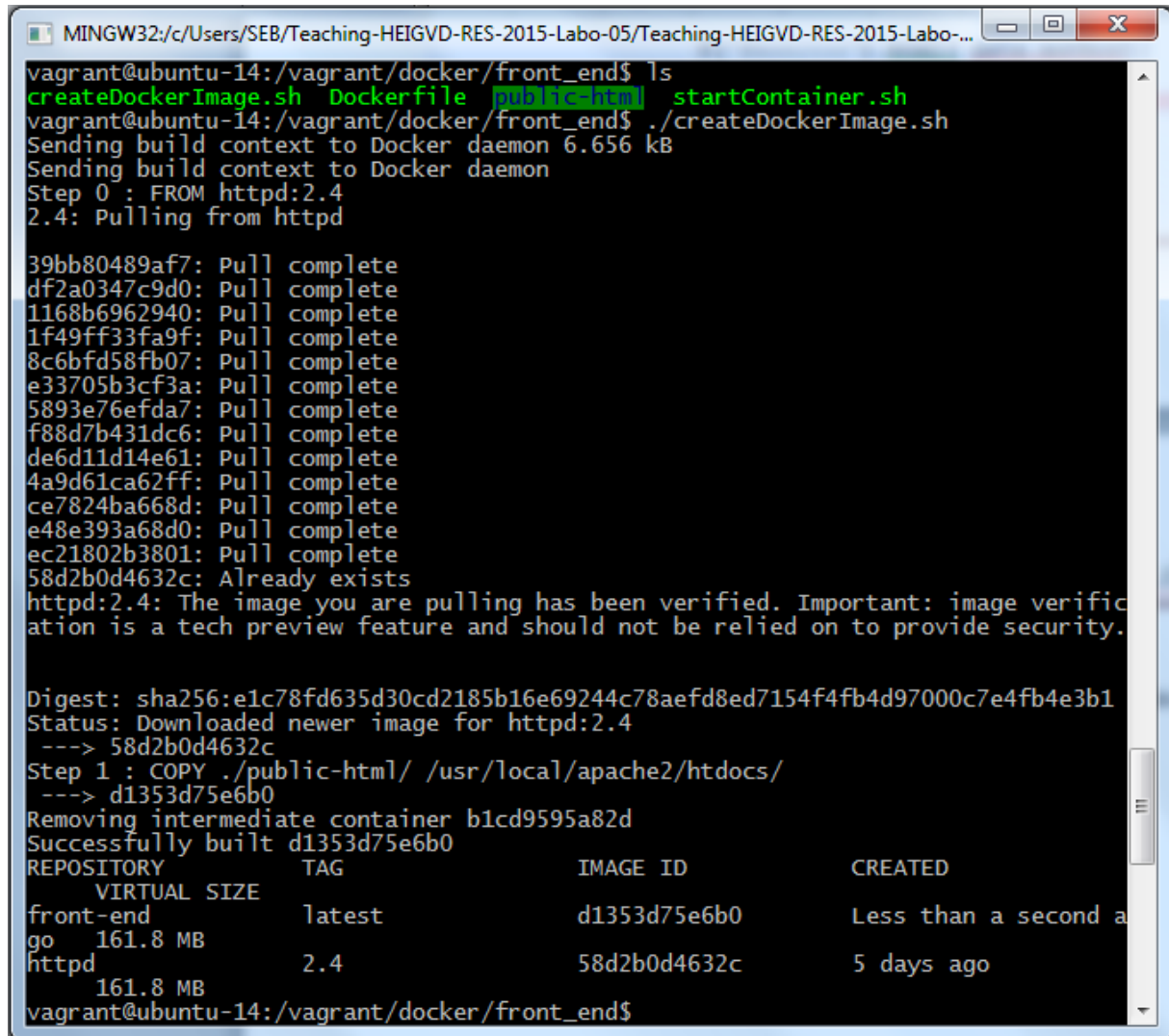
```
<script language="JavaScript">
```

```
$(document).ready(function () {
    refreshNodes();
});
function refreshNodes() {
    $.getJSON('/api/message',
        function(data) {

            $('#monitor').html( data.author);
        });
    var t=setTimeout("refreshNodes()", 1000);
}
</script>
```

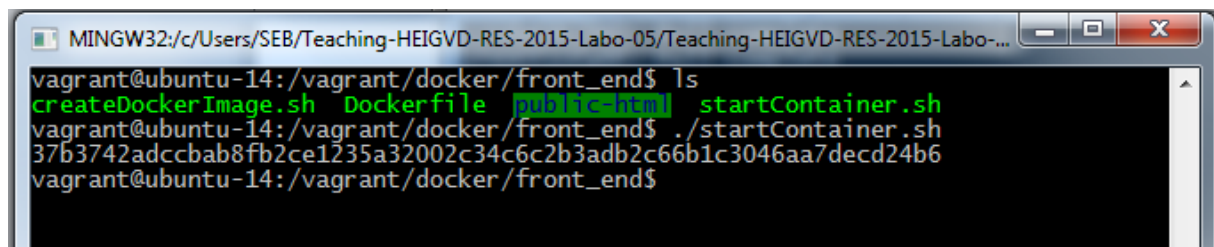
On peut ensuite démarrer un container grâce au deux scripts créés précédemment :

La création de l'image prend quelques minutes.

A terminal window titled 'MINGW32:/c/Users/SEB/Teaching-HEIGVD-RES-2015-Labo-05/Teaching-HEIGVD-RES-2015-Labo-...' shows the execution of 'createDockerImage.sh'. The script pulls the 'httpd:2.4' image from Docker Hub, verifies it, and then builds a new image 'd1353d75e6b0' from the 'public-html' Dockerfile. The terminal output includes a list of layers being pulled, a verification message from Docker, and a table of the resulting images.

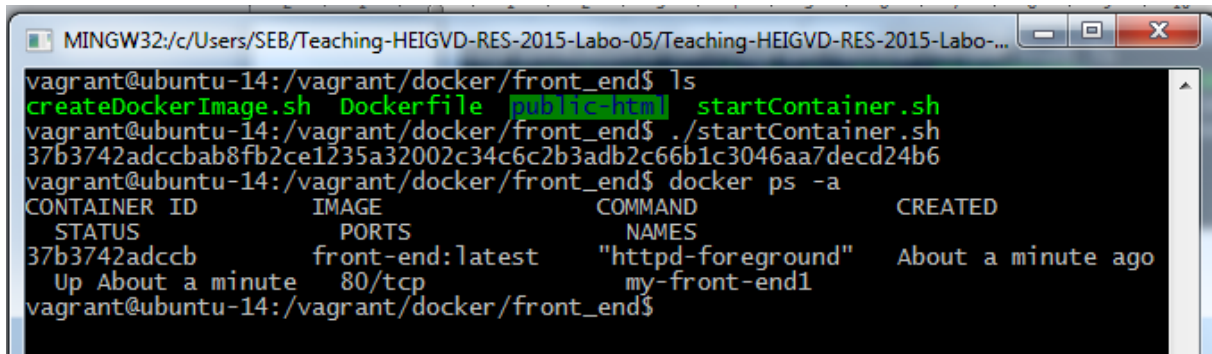
```
vagrant@ubuntu-14:/vagrant/docker/front_end$ ls
createDockerImage.sh Dockerfile public-html startContainer.sh
vagrant@ubuntu-14:/vagrant/docker/front_end$ ./createDockerImage.sh
Sending build context to Docker daemon 6.656 kB
Sending build context to Docker daemon
Step 0 : FROM httpd:2.4
2.4: Pulling from httpd
39bb80489af7: Pull complete
df2a0347c9d0: Pull complete
1168b6962940: Pull complete
1f49ff33fa9f: Pull complete
8c6bfd58fb07: Pull complete
e33705b3cf3a: Pull complete
5893e76efda7: Pull complete
f88d7b431dc6: Pull complete
de6d11d14e61: Pull complete
4a9d61ca62ff: Pull complete
ce7824ba668d: Pull complete
e48e393a68d0: Pull complete
ec21802b3801: Pull complete
58d2b0d4632c: Already exists
httpd:2.4: The image you are pulling has been verified. Important: image verification is a tech preview feature and should not be relied on to provide security.
Digest: sha256:e1c78fd635d30cd2185b16e69244c78aefd8ed7154f4fb4d97000c7e4fb4e3b1
Status: Downloaded newer image for httpd:2.4
---> 58d2b0d4632c
Step 1 : COPY ./public-html/ /usr/local/apache2/htdocs/
---> d1353d75e6b0
Removing intermediate container b1cd9595a82d
Successfully built d1353d75e6b0
REPOSITORY          TAG                 IMAGE ID            CREATED
VIRTUAL SIZE
front-end            latest             d1353d75e6b0       Less than a second ago
go                   161.8 MB
httpd                2.4               58d2b0d4632c       5 days ago
                    161.8 MB
vagrant@ubuntu-14:/vagrant/docker/front_end$
```

Puis, on démarre un container avec cette image créée :

A terminal window titled 'MINGW32:/c/Users/SEB/Teaching-HEIGVD-RES-2015-Labo-05/Teaching-HEIGVD-RES-2015-Labo-...' shows the execution of 'startContainer.sh'. The script runs 'docker run' to start a container from the 'd1353d75e6b0' image, displaying the container ID '37b3742adccbab8fb2ce1235a32002c34c6c2b3adb2c66b1c3046aa7decd24b6'.

```
vagrant@ubuntu-14:/vagrant/docker/front_end$ ls
createDockerImage.sh Dockerfile public-html startContainer.sh
vagrant@ubuntu-14:/vagrant/docker/front_end$ ./startContainer.sh
37b3742adccbab8fb2ce1235a32002c34c6c2b3adb2c66b1c3046aa7decd24b6
vagrant@ubuntu-14:/vagrant/docker/front_end$
```

On vérifie que tout s'est bien passé :

A terminal window titled 'MINGW32:/c/Users/SEB/Teaching-HEIGVD-RES-2015-Labo-05/Teaching-HEIGVD-RES-2015-Labo-...' shows the following commands and output:

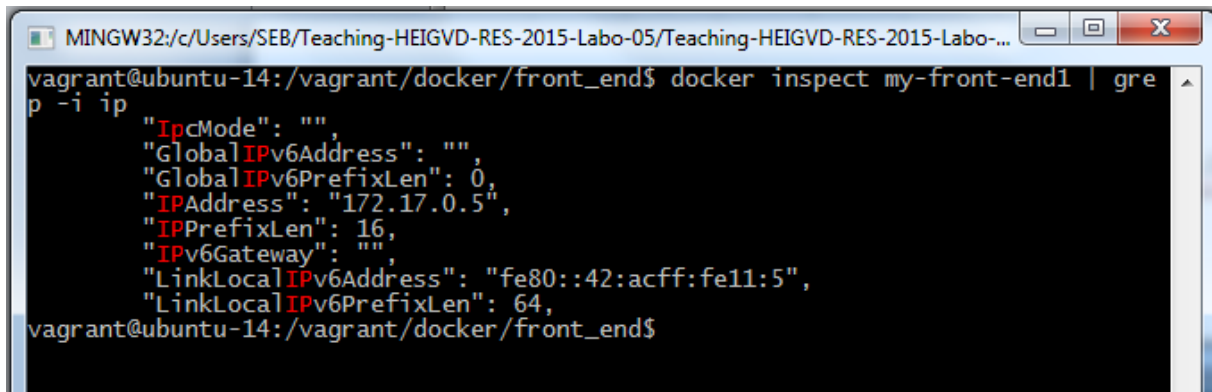
```
vagrant@ubuntu-14:/vagrant/docker/front_end$ ls
createDockerImage.sh Dockerfile public-html startContainer.sh
vagrant@ubuntu-14:/vagrant/docker/front_end$ ./startContainer.sh
37b3742adccbab8fb2ce1235a32002c34c6c2b3adb2c66b1c3046aa7decd24b6
vagrant@ubuntu-14:/vagrant/docker/front_end$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED
37b3742adccb	front-end:latest	"httpd-foreground"	About a minute ago
Up About a minute	80/tcp	my-front-end1	

```
vagrant@ubuntu-14:/vagrant/docker/front_end$
```

Tout semble ok, le container existe.

On va finalement récupérer l'adresse ip du container pour pouvoir paramétrer le serveur à la fin :

A terminal window titled 'MINGW32:/c/Users/SEB/Teaching-HEIGVD-RES-2015-Labo-05/Teaching-HEIGVD-RES-2015-Labo-...' shows the following command and output:

```
vagrant@ubuntu-14:/vagrant/docker/front_end$ docker inspect my-front-end1 | grep -i ip
```

```
"IpcMode": "",
"GlobalIPv6Address": "",
"GlobalIPv6PrefixLen": 0,
"IPAddress": "172.17.0.5",
"IPPrefixLen": 16,
"IPv6Gateway": "",
"LinkLocalIPv6Address": "fe80::42:acff:fe11:5",
"LinkLocalIPv6PrefixLen": 64,
```

```
vagrant@ubuntu-14:/vagrant/docker/front_end$
```

Le container implémentant le front-end possède l'adresse ip 172.17.0.5.

### 3 Mise en place du back-end

Nous avons utilisé l'image nommée « image\_express.js » du labo3.

Tout d'abord, on crée l'image (capture coupée) :

```

MINGW32/c/Users/SEB/Teaching-HEIGVD-RES-2015-Labo-05/Teaching-HEIGVD-RES-2015-Labo-...
vagrant@ubuntu-14:/vagrant/docker$ ls
front_end  httpd-reverse-proxy  image_expressjs  image_nodejs  shared_volume
vagrant@ubuntu-14:/vagrant/docker$ docker build -t back-end image_expressjs/
Sending build context to Docker daemon 20.48 kB
Sending build context to Docker daemon
Step 0 : FROM node:0.10.38
0.10.38: Pulling from node
7a3871ba15f8: Pull complete
a2703ed272d7: Pull complete
c9e3effdd23a: Pull complete
7711db4bb553: Pull complete
c37027914114: Pull complete
0923f3bbc71e: Pull complete
b5f28cb2de41: Pull complete
79b27c4ea3bc: Already exists
39bb80489af7: Already exists
df2a0347c9d0: Already exists
node:0.10.38: The image you are pulling has been verified. Important: image verification is a tech preview feature and should not be relied on to provide security.

Digest: sha256:865dbc9489b9fadc0264de82a9beecc6e6eec4fcce89930bec0b066c41211dc1
Status: Downloaded newer image for node:0.10.38
----> 79b27c4ea3bc
Step 1 : MAINTAINER Olivier Liechti
----> Running in ad3c146be1b5
----> 428c0f157d39

```

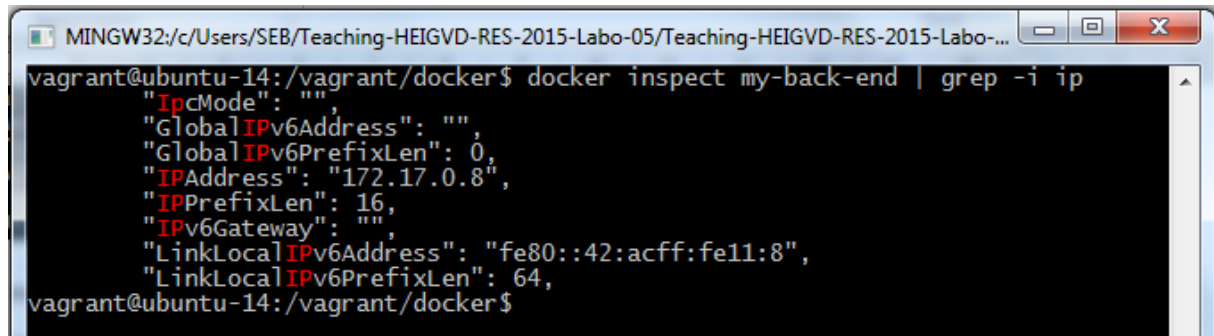
Ensuite, on lance un container avec cette image, et on vérifie que tout s'est déroulé correctement :

```

MINGW32:/c:/Users/SEB/Teaching-HEIGVD-RES-2015-Labo-05/Teaching-HEIGVD-RES-2015-Labo-...
vagrant@ubuntu-14:/vagrant/docker$ docker run -d --name my-back-end back-end
87e7104e2d78dbea6a34ca96c34ab7564767cf62aaee0b16253dc17d70085b0a
vagrant@ubuntu-14:/vagrant/docker$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS            PORTS              NAMES
87e7104e2d78       back-end:latest    "/bin/sh -c 'pm2 sta" 4 seconds ago
Up 4 seconds                               my-back-end
37b3742adcdb       front-end:latest   "httpd-foreground"    27 minutes ago
Up 27 minutes      80/tcp             my-front-end1
vagrant@ubuntu-14:/vagrant/docker$

```

On récupère l'adresse IP :



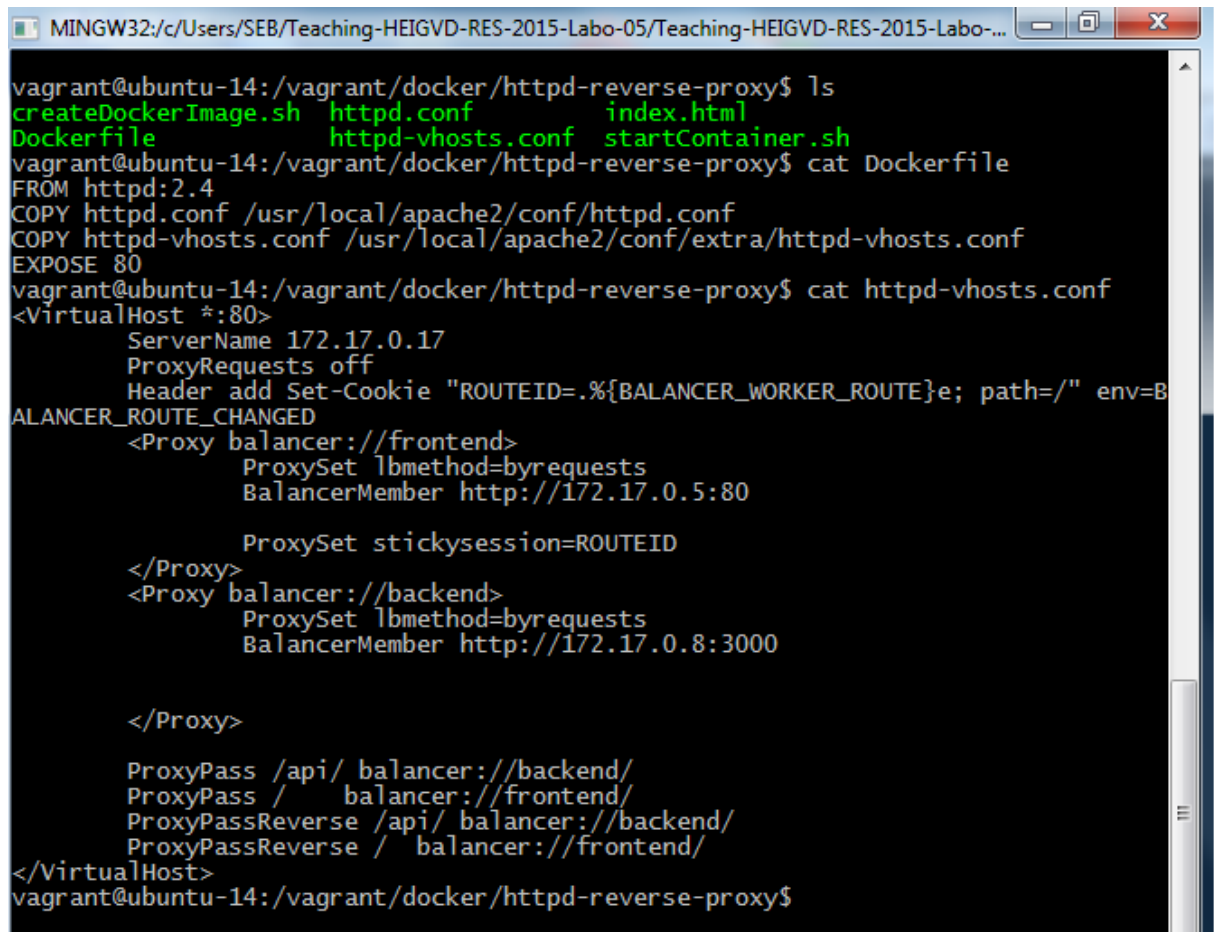
```
MINGW32:/c/Users/SEB/Teaching-HEIGVD-RES-2015-Labo-05/Teaching-HEIGVD-RES-2015-Labo-...
vagrant@ubuntu-14:/vagrant/docker$ docker inspect my-back-end | grep -i ip
    "IpMode": "",
    "GlobalIPv6Address": "",
    "GlobalIPv6PrefixLen": 0,
    "IPAddress": "172.17.0.8",
    "IPPrefixLen": 16,
    "IPv6Gateway": "",
    "LinkLocalIPv6Address": "fe80::42:acff:fe11:8",
    "LinkLocalIPv6PrefixLen": 64,
vagrant@ubuntu-14:/vagrant/docker$
```

L'adresse ip du container implémentant le back-end est 172.17.0.8



#### 4 Mise en place du serveur (reverse proxy + load-balancer)

On utilise la même image httpd que précédemment pour le front-end :



```
MINGW32/c/Users/SEB/Teaching-HEIGVD-RES-2015-Labo-05/Teaching-HEIGVD-RES-2015-Labo-...
vagrant@ubuntu-14:/vagrant/docker/httpd-reverse-proxy$ ls
createDockerImage.sh  httpd.conf  index.html
Dockerfile            httpd-vhosts.conf  startContainer.sh
vagrant@ubuntu-14:/vagrant/docker/httpd-reverse-proxy$ cat Dockerfile
FROM httpd:2.4
COPY httpd.conf /usr/local/apache2/conf/httpd.conf
COPY httpd-vhosts.conf /usr/local/apache2/conf/extra/httpd-vhosts.conf
EXPOSE 80
vagrant@ubuntu-14:/vagrant/docker/httpd-reverse-proxy$ cat httpd-vhosts.conf
<VirtualHost *:80>
    ServerName 172.17.0.17
    ProxyRequests off
    Header add Set-Cookie "ROUTEID=.%{BALANCER_WORKER_ROUTE}e; path=/" env=BALANCER_ROUTE_CHANGED
    <Proxy balancer://frontend>
        ProxySet lbmethod=byrequests
        BalancerMember http://172.17.0.5:80

        ProxySet stickysession=ROUTEID
    </Proxy>
    <Proxy balancer://backend>
        ProxySet lbmethod=byrequests
        BalancerMember http://172.17.0.8:3000

    </Proxy>

    ProxyPass /api/ balancer://backend/
    ProxyPass / balancer://frontend/
    ProxyPassReverse /api/ balancer://backend/
    ProxyPassReverse / balancer://frontend/
</VirtualHost>
vagrant@ubuntu-14:/vagrant/docker/httpd-reverse-proxy$
```

Grâce aux adresses ip récupérée, on peut configurer correctement le load-balancing.

Quand le serveur verra passer une requête pour /api, il saura qu'il doit la transmettre au back-end, et si c'est plutôt / alors il sait que cela concerne le front-end.

Et le front-end utilise les sticky-sessions.

On peut donc créer l'image :

```

MINGW32/c/Users/SEB/Teaching-HEIGVD-RES-2015-Labo-05/Teaching-HEIGVD-RES-2015-Labo-...
</html>
vagrant@ubuntu-14:/vagrant/docker/httpd-reverse-proxy$ ls
createDockerImage.sh  httpd.conf  index.html
Dockerfile            httpd-vhosts.conf  startContainer.sh
vagrant@ubuntu-14:/vagrant/docker/httpd-reverse-proxy$ ./createDockerImage.sh
Sending build context to Docker daemon 179.2 kB
Sending build context to Docker daemon
Step 0 : FROM httpd:2.4
--> 58d2b0d4632c
Step 1 : COPY httpd.conf /usr/local/apache2/conf/httpd.conf
--> 043f43cfd911
Removing intermediate container 31750cbd1ea3
Step 2 : COPY httpd-vhosts.conf /usr/local/apache2/conf/extra/httpd-vhosts.conf
--> 04354120212b
Removing intermediate container f659b4b71287
Step 3 : EXPOSE 80
--> Running in b45445c70f0a
--> b3dc09eb6d0e
Removing intermediate container b45445c70f0a
Successfully built b3dc09eb6d0e
REPOSITORY          TAG          IMAGE ID          CREATED
VIRTUAL SIZE
apache-reverse-proxy latest       b3dc09eb6d0e      Less than a second ago
161.9 MB
back-end            latest       4ff5be246ddc      17 minutes ago
719.2 MB
front-end           latest       d1353d75e6b0      45 minutes ago
161.8 MB
node                0.10.38     79b27c4ea3bc      3 days ago
702.2 MB
httpd               2.4         58d2b0d4632c      5 days ago
161.8 MB
vagrant@ubuntu-14:/vagrant/docker/httpd-reverse-proxy$

```

On crée le container associé et on vérifie que tout s'est bien passé :

```

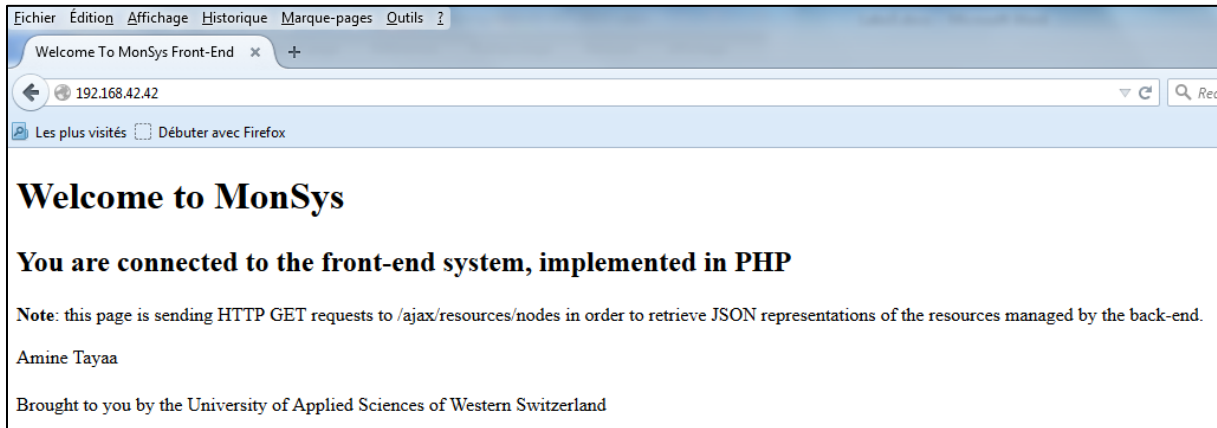
vagrant@ubuntu-14:/vagrant/docker/httpd-reverse-proxy$ ./startContainer.sh
39d05db781fa502fc3da9410a4c3df6675d1a5796deb593289b163fbb195dd76
vagrant@ubuntu-14:/vagrant/docker/httpd-reverse-proxy$ docker ps -a
CONTAINER ID        STATUS          IMAGE                PORTS                COMMAND              NAMES              CREATED
39d05db781fa        Up 10 seconds   apache-reverse-proxy:latest   0.0.0.0:80->80/tcp   "httpd-foreground"   my-reverse-proxy-apache   11 seconds ago
87e7104e2d78        Up 15 minutes   back-end:latest           "/bin/sh -c 'pm2 sta   my-back-end         15 minutes ago
37b3742adccb        Up 42 minutes   front-end:latest          80/tcp              "httpd-foreground"   my-front-end1         42 minutes ago
vagrant@ubuntu-14:/vagrant/docker/httpd-reverse-proxy$

```

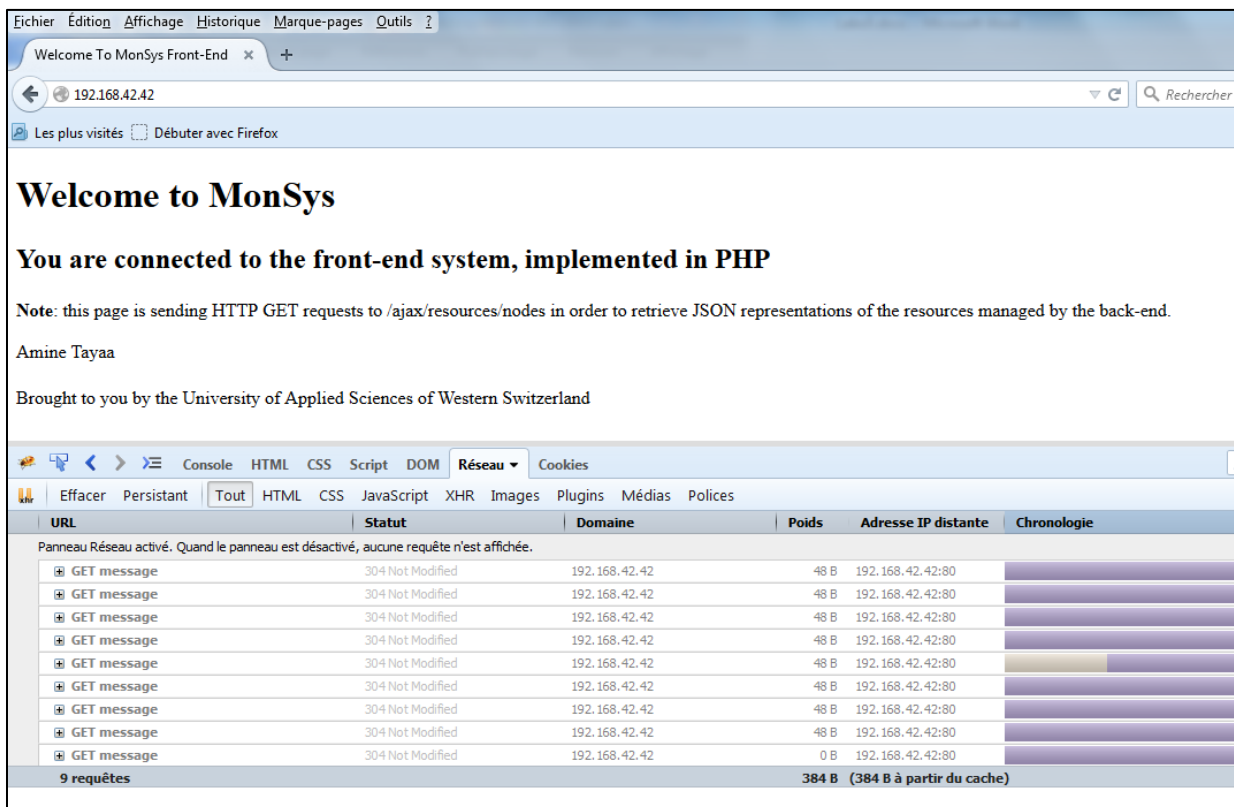
Donc on a bien nos trois containers : le front-end, le back-end et le serveur.

## 5 Tests

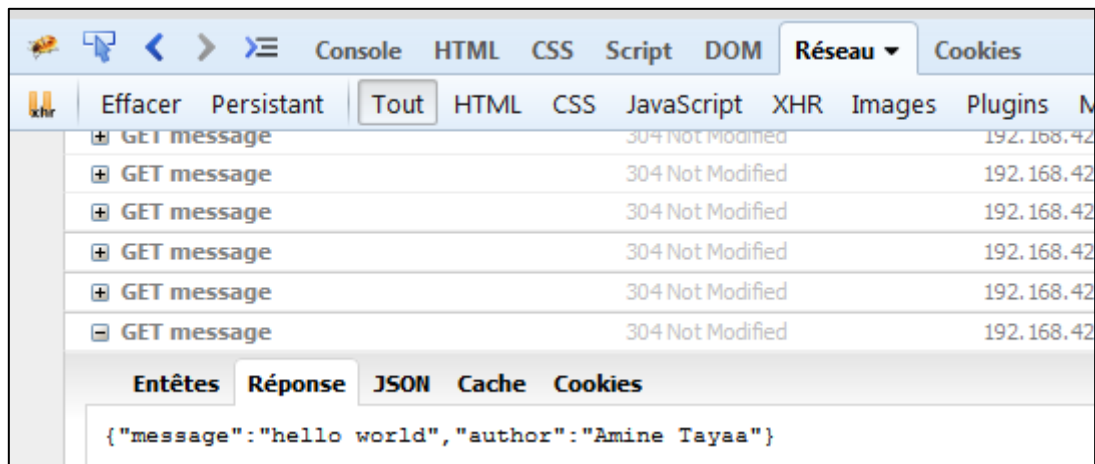
Testons déjà l'ouverture de la page web à l'adresse 192.168.42.42 (adresse configurée statiquement dans le fichier Vagrantfile) :



Et si on regarde avec fireBug, on devrait voir les requêtes GET toutes les secondes :



On voit que la réponse concerne bien la représentation JSON du message et de l'auteur :



Le contenu délivré par le back-end n'est pas dynamique pour le moment.

Pour ce faire, l'idée est d'afficher sur la page web, les adresses ip des containers répondant. Par conséquent, on verra si les sticky-sessions sont fonctionnels pour le front-end et pas pour le back-end.

## 6 Conclusion

Le dialogue s'effectue correctement entre le serveur, le front-end et le back-end, mais lorsqu'il existe seulement un exemplaire de chaque image.

Nous n'avons pas réussi à implémenter les sticky-sessions (en ajoutant route=1, route=2).

Nous nous sommes pris beaucoup trop tard pour ce laboratoire et nous nous sommes laissé submergé par la charge de travail.

Nous avons mis la priorité sur un autre cours.

Yverdon-les-Bains, le 1 juin 2015

Amine Tayaa – Valentin Schaad

Sébastien Henneberger – (Benoît Zuckschwerdt)