11/06/2017

Laboratoire : RES

Infrastructure http

wojtek myszkorowski et Jérémie zanone
HEIG-VD

Table des matières

Introduction

In this practical assignment, we want to setup a reverse proxy that will allow us to connect to different containers that and will allows to create a static page with dynamic content. We will need to write some JavaScript code, and be able to extend our knowledge to web infrastructure. We will learn some new concept such the famous "same origin policy". We will also use some basic concept about JQery and Ajax.

First assignment

In this first assignment, the goal is to get familiar with an apache server. We use images from Docker to display web pages.

First of all, we need to get some information about our environment we use the command to know our ipadress " ifconfg " we search for our ip4 address. In my case, it is the 192.168.0.104 I used this ipaddess because I am on linux but I can use the docker addres which is 172.17.0.1 After this we can download 2 things:

1. is an php image
2. is bootstrap template that we will display

We start by building a Dockefile that will contain the minimum for running. Like a *From* instruction and *COPY* to run a first container. To run for the first time we use the following command

```
docker run -d -p 9090:80 php:7.0-apache
```

we can use

```
docker logs
```

to check if we have some errors once the container has been started.
To test properly we can use the telnet command on our docker file

```
telnet 192.168.0.104 9090
```

and make *a GET /HTTP/1.0* request and we have an answer.

Once we checked that our Dockefile is functioning we can copy the bootstrap files into our content directory. Change the name of the directory from *src* to *content*

```
docker inspect -it name_Image /bin/bash
```

this allow us to get inside the image and see what kind of files it contains.

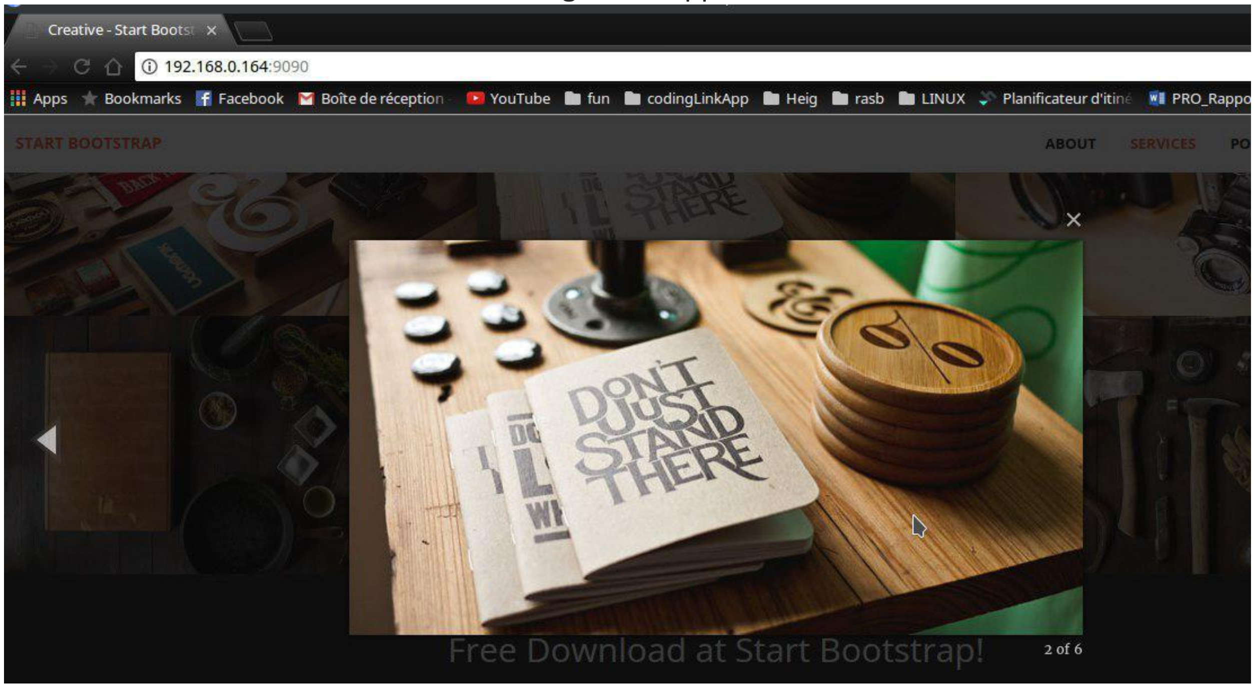
```
docker build -t res/apache_php .
```

this command allows us to build an image that will use our content which contains files to display the template web site.

```
docker run -p 9090:80 res/apache_php
```

this command runs the images and allows us to connect on our web brower to the address : 192.168.0.104:9090 and we can see the display of the web template that we have downloaded.

As we want to check the correct functioning of the application



this web page has been modified ant it shows that we are able to modify an html page.

Second assignment

In this part we want to create a dynamic JavaScript application that we will build some random data.
we need on a official image like that node to create our container. We can find it on dockerHub.
As in the first assignmenent we need to implement our Dockerfile, based on the same instructions

`npm init` this is for preparing the JSON package with information about our image.
`npm install --save chance` this is for saving the dependencies for the chance library

After we can build our image and run it and it will display some names inside the terminal. If we execute the bin/bash command and inspect the /opt/app files it will contain everything that we had inside our file named *src*, it has made a copy of everything from this location to /opt/app.

In a second time, we are supposed to create an app that sends information and we must listen from a port and retrieve that information

`npm install --save express` this allows us to install the dependencies that uses random numbers

we can launch the application by using the command "*node"* and then the command "*telnet localhost*" to connect to the application and interact with it.

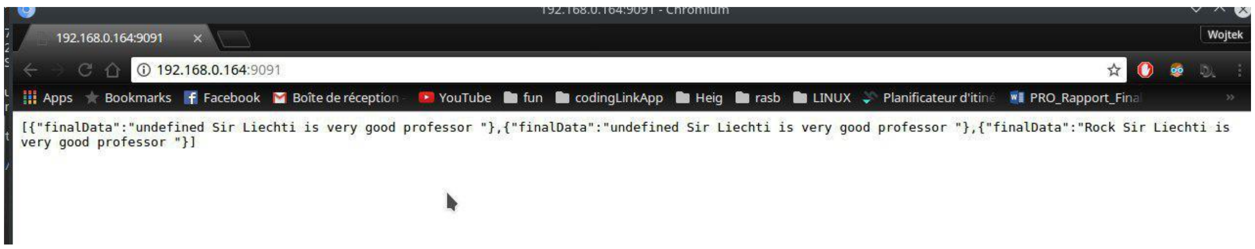
`npm install random-js` we have used this library to generate random information as asked.
Everytime that we want to use a certain library we some functionalities we have to install it by using the command "npm".

Our code is available on the github.
Now that our index.js is ready and our Dockerfile completed we can start to build our image.

```
docker build -t res/express_partie2 .
```

```
docker run -p 9091:3000 res/express_partie2
```

The code can be seen in our github. The link is at the end of the report



```
[{"finalData":"undefined Sir Liechti is very good professor "},{"finalData":"undefined Sir Liechti is very good professor "},{"finalData":"Rock Sir Liechti is
very good professor "}]
```

We can use Postman as well to check the functioning of our setup. In this case since we have only 2-3 requests in total. We did not use it.

Third assignment

In this third assignment, we want to setup a reverse proxy to be able from a browser to contact our containers according to the information we are looking for.
In this precise situation we want to communicate easily with our containers. That is why we need this entry Point which will be our reverse proxy container.
First we need to know the ip address of our containers we use the following command for that

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

```
docker run -d –name express_dynamic res/express_partie2
```

```
docker inspect apache_static | grep -I ipaddress
```

gives 172.17.0.2

```
docker inspect express_dynamic | grep -I ipaddress
```

gives 172.17.0.3

we uses the "telnet" function with right port mapping 80 and 3000 for the second. We can access the data inside our containers and retrieve information with a GET.

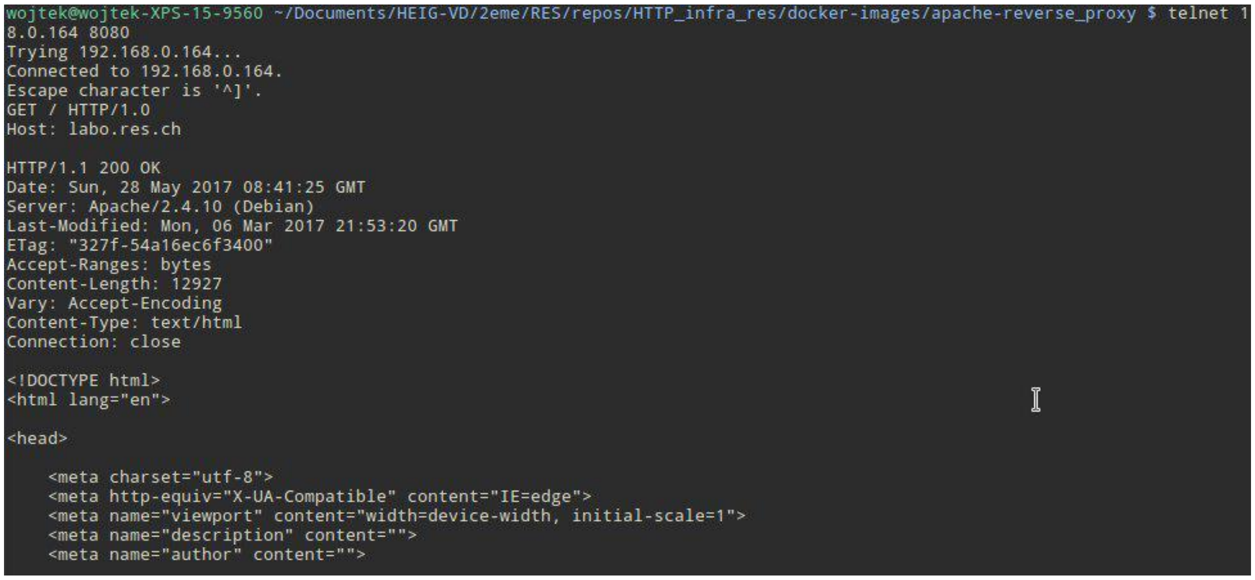
```
docker run -it -p 8080:80 php:7.0-apache /bin/bash
```

after this we are going to edit the content of the file .conf that we have copied from the original

```
cp 000-default.conf 001-reverse-proxy.conf
```

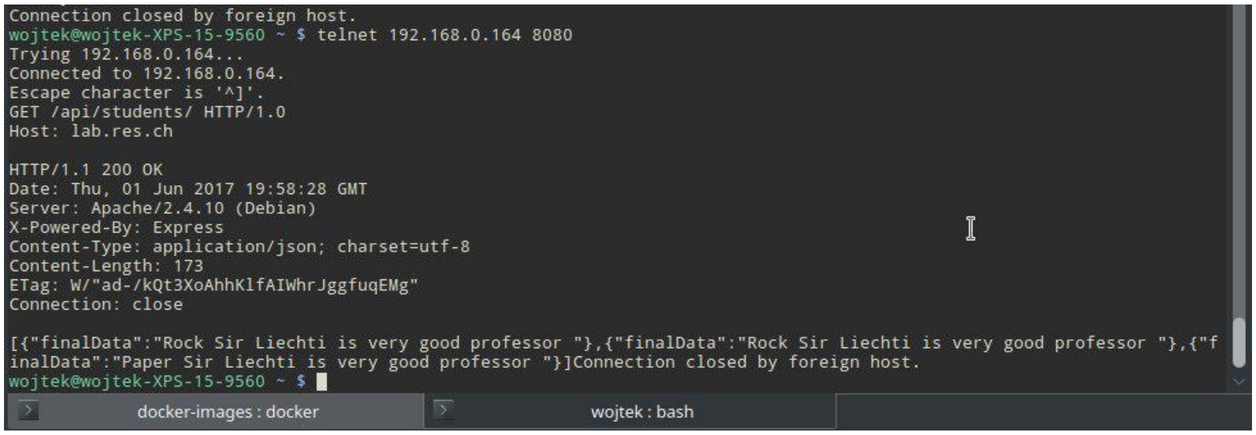
Important is to activate to the modules in order to be able to run the config proxy
"a2enmod proxy"
"a2enmod proxy_http"

So its working for more general rule.



```
wojtek@wojtek-XPS-15-9560 ~/Documents/HEIG-VD/2eme/RES/repos/HTTP_infra_res/docker-images/apache-reverse_proxy $ telnet 1
8.0.164 8080
Trying 192.168.0.164...
Connected to 192.168.0.164.
Escape character is '^]'.
GET / HTTP/1.0
Host: labo.res.ch

HTTP/1.1 200 OK
Date: Sun, 28 May 2017 08:41:25 GMT
Server: Apache/2.4.10 (Debian)
Last-Modified: Mon, 06 Mar 2017 21:53:20 GMT
ETag: "327f-54a16ec6f3400"
Accept-Ranges: bytes
Content-Length: 12927
Vary: Accept-Encoding
Content-Type: text/html
Connection: close

<!DOCTYPE html>
<html lang="en">

<head>

    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="description" content="">
    <meta name="author" content="">
```

And for the dynamic server



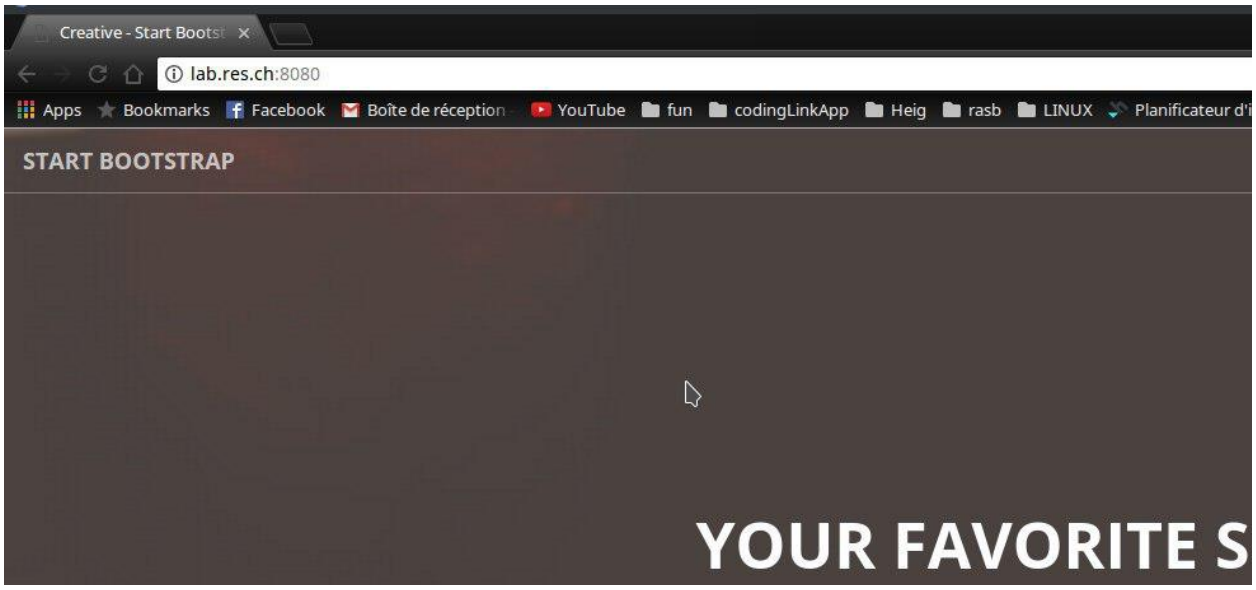
```
Connection closed by foreign host.
wojtek@wojtek-XPS-15-9560 ~ $ telnet 192.168.0.164 8080
Trying 192.168.0.164...
Connected to 192.168.0.164.
Escape character is '^]'.
GET /api/students/ HTTP/1.0
Host: lab.res.ch

HTTP/1.1 200 OK
Date: Thu, 01 Jun 2017 19:58:28 GMT
Server: Apache/2.4.10 (Debian)
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 173
ETag: W/"ad-/kQt3XoAhhKlfAIWhrJggfuqEMg"
Connection: close

[{"finalData":"Rock Sir Liechti is very good professor "},{"finalData":"Rock Sir Liechti is very good professor "},{"f
inalData":"Paper Sir Liechti is very good professor "}]Connection closed by foreign host.
wojtek@wojtek-XPS-15-9560 ~ $
```

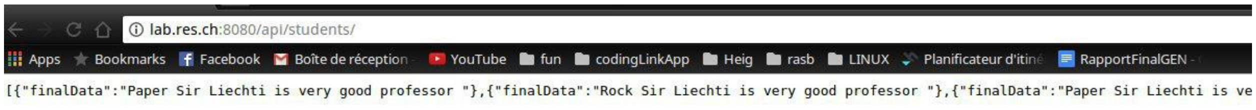
Both connection works although I need to specify the host in the second command.

In a second part of this assignment we have to do it by setting up the container which makes the links.

We need first to build our new images

```
docker build -t res/apache_rp .
```





```
[{"finalData":"Paper Sir Liechti is very good professor "},{"finalData":"Rock Sir Liechti is very good professor "},{"finalData":"Paper Sir Liechti is ve
```

And Voila for the second part of this video.

In the third part we do the exact same thing, only this time we create our container and put the code inside. First we need to create the config files such as the 000-default.conf and the 001-reverse-proxy.conf which contains our protocol and configuration. In the Dockerfile as seen before we need to activate the following modules proxy and proxy_http and a2ensite to active the virtualhosts.
The protocol can be found inside the Dockerfile on the github.

We have to run our 3 images now.
the first two are the commands seen in the previous part

```
docker run -d –name apache_static res/apache_php
docker run -d –name express_dynamic res/express_partie2
```

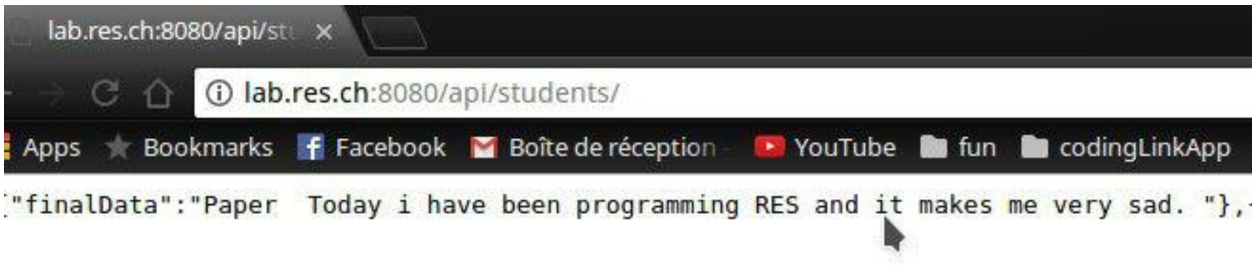
The third one is

```
docker run -p 8080:80 res/apache_rp
```

If we try to access it from the terminal it gives us the same result as seen previously. Now we want to access these 2 containers from our browser.
we need to configure the file /etc/hosts which will allows us define our DNS address of our container.
so we add the following line to our hosts file

```
192.168.0.104 lab.res.ch
```

And now we can access it on the following address lab.res.ch:8080.



```
"finalData":"Paper  Today i have been programming RES and it makes me very sad. "},
```

Here is the screenshot.

Fourth assignment

In this fourth assignment, we want to create JQuery code to make ajax request to our dynamic container in order to send data to our web page.
To proceed there is only a few steps to do. We must start our 3 containers in the right order because of the IP address given.
First, we start as in the previous assignment the apache_static then the dynamic. Finally, the apache_proxy with a port mapping “-p 8080:80” in order to be able to communicate from our browser.

As we wish to change some HTML content we have to use JQuery to recover the tag that we want to change. To proceed we use a function that recovers the tag and changes to the data that we have requested from our dynamic container.
This function will go inside our apache-php image. We create a new file with the JavaScript code. And we need to add the resources to our index.html in order to call it when we the page is accessed.
The configuration is to be done inside our content of the apache_static image. We need to add a new file with the function that will retrieve the data and change it. We need to tell the index.html to look for this JavaScript code. So, we add a script at the end of the index.html file.

In order to make permeant we have to add this files into our repository so that when we build our image then the code is running instantly.
For this to work it is very important to start the containers in the right order. Because we have hardcoded the ipaddresses. First we start the static_php container then the dynamic and finally the proxy.

Here is the code that allows this assignment to function correctly.

```
$(function() {
    console.log("loading data");

    function loadData() {
      $.getJSON("/api/students/", function (data){
        console.log(data);
        var message = "we are the five guys";
        if(data.length > 0) {

            message = data[0].finalData;

        }
        $(".section-heading").text(message);
      });
    };
    loadData();
    setInterval(loadData, 2000);
});
```

I used this code. The first line allows me to start my giving a callback function that is called by the variable $ which is shortcut for when the library is available then execute the function. Then we have our main function which loadData() that will load the data from the dynamic container. When it receives information, it executes the next callback function. That will display inside my web page the data I want.

Fifth assignment

In this last assignment, we want to get rid of the hard-coded IP addresses that are in our containers. Specially the hard-coded IP addresses inside our web proxy which requires every time that we start our containers in a precise order.

We start by creating a new file next to our dockerfile in the apache-reverse directory. The new file is apache2-foreground to this file we have to gives the rights in order to execute

```
chmod 755 apache2-foreground
```

This a script that will allow us to see if our environment variables do have the right values once our container is started with the -e option.
and we need to copy that new file inside our image once it has been constructed.
COPY apache2-foreground /usr/local/bin/

This is the content of the file apache2-foreground. It will allows to pass environment variable when we start the container.

Now we need to setup a new file which will allow us to build the setup that will allow our php interpreter to contact the right containers. First we create a new file named *config-template.php*. This file will allow to match our variable of environment to our proxy-reverse container.

The code can be found in our repository at *https://github.com/CoolPolishGuy/HTTP_infra_res/blob/master/docker-images/apache-reverse_proxy/templates/config-template.php*.
We need also to add a new line to our docker which will allow to the container to use this script. Therefore we copy the template like this *COPY templates /var/apache2/templates*

Now that our structure is ready. We can start to test it .

The last command that we run are

```
docker run -d res/apache_php
```

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

```
docker run -d res/express_partie2
```

```
docker run -d —name express_dynamic res/express_partie2
```

We can check the IP address by using the command "docker inspect express_dynamic | -I ipaddress" which allows us to verifiy each address that we need.

```
docker run -d -e STATIC_APP=172.17.0.3:80 -e DYNAMIC_APP=172.17.0.5:3000 —
name apache_rp -p 8080:80 res/apache_rp
```

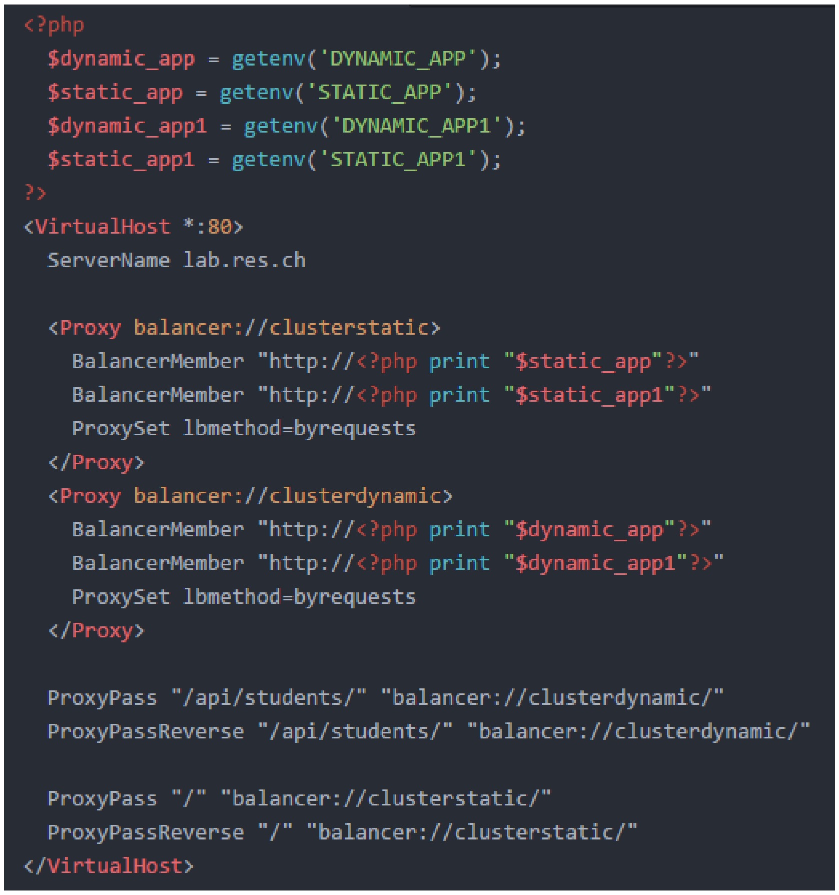
We can always test our setup by using commands like this one "docker exec -it nameOfContainer /bin/bash" this will allow us to go inside our container and verify that it contains every file we have copied

Bonus Balancing

We have implemented the first bonus part. In this part, what we want is to be able to start a few containers and affect them to variables. These variables are connected into our proxy. Therefore, if we kill one and that there is one similar container left. Then it will continue to work as previously.
Our setup only needs a few modifications of it to work.

We need to modify 2 files mainly the dockerfile and our config-template.php
Inside our Dockerfile we need to add 2 things. We need to enable modules that will allow the balancing between containers and the mod of balancing. These two things are done by using this line

"proxy_balancer lbmethod_byrequests" this line is now added to a2enmod.
And here is our code for the config php



```
<?php
  $dynamic_app = getenv('DYNAMIC_APP');
  $static_app = getenv('STATIC_APP');
  $dynamic_app1 = getenv('DYNAMIC_APP1');
  $static_app1 = getenv('STATIC_APP1');
?>
<VirtualHost *:80>
  ServerName lab.res.ch

  <Proxy balancer://clusterstatic>
    BalancerMember "http://<?php print "$static_app"?>"
    BalancerMember "http://<?php print "$static_app1"?>"
    ProxySet lbmethod=byrequests
  </Proxy>
  <Proxy balancer://clusterdynamic>
    BalancerMember "http://<?php print "$dynamic_app"?>"
    BalancerMember "http://<?php print "$dynamic_app1"?>"
    ProxySet lbmethod=byrequests
  </Proxy>

  ProxyPass "/api/students/" "balancer://clusterdynamic/"
  ProxyPassReverse "/api/students/" "balancer://clusterdynamic/"

  ProxyPass "/" "balancer://clusterstatic/"
  ProxyPassReverse "/" "balancer://clusterstatic/"
</VirtualHost>
```

There is not much to say about. We have declared four variables and we assigne them inside our clusters. Each time we put the 2 variables inside. We could have more than just 2, we could have declared 3-4-20 variables if wanted. But the principle stays the same.

Now how do we launch it?
Well we do exactly what we have been doing during thoses previous part.
We start by build images.

```
docker build -t res/apache_php .
```

```
docker build -t res/expresspartie2 .
```

```
docker build -t res/apache_rp .
```

We can run them now

```
docker run -d res/apache_php
```

```
docker run -d res/apache_php
```

```
docker run -d res/expresspartie2
docker run -d res/expresspartie2
```

We don’t need to give them names. But we could if want to make things easier once we want to kill them.
Now we start our proxy

```
Docker run -d -e STATINC_APP=172.17.0.2:80 -e STATINC_APP1=172.17.0.3:80 -e
DYNAMIC_APP=172.17.0.4:3000 -e DYNAMIC_APP1=172.17.0.5:3000 –name apache_rp -p 8080:80
res/apache_rp
```

What is important is to verify that we started each container, that we affect them to correct variable. If we are not sure about the ip address we just type this command.

```
Docker inspect nameOFcontainer | grep -i ipaddr
```

Now we can say that everything is running as it should, (browser address at lab.res.ch:8080)
We can start to kill a container to see the balancing.

We use `docker ps` to get the name of the container we want to kill. And then we use

```
Docker kill (nameOFcontainer)
```

AND VOILA.

Conclusion

To see our code you can go the following address : https://github.com/CoolPolishGuy/HTTP_infra_res

This practical assignment shows us the complexity of the infrastructure of the web. But it showed also the strength of the tools that we can use in order to do different things. We have learned how to debug some code with many different tools like dev tool from firefox, telnet command and so on. We can say that this assignment was one the hardest because we don’t have much knowledge into the web concept. We have spent much time debugging the first part of the bonus, which didn’t allow us to do more bonus assignment unfortunately. We have also learned a bit about jQuery and ajax and discover how simple it is to modify some html content.