REPORT Teaching-HEIGVD-RES-2017-Labo-HTTPInfra

Authors : Tchente Dany and Luca Sivillica

Step 1: Static HTTP server with apache httpd

Acceptance criteria

- You have a GitHub repo with everything needed to build the Docker image.
- You do a demo, where you build the image, run a container and access content from a browser.
- You have used a nice looking web template, different from the one shown in the webcast.
- You are able to explain what you do in the Dockerfile.
- You are able to show where the apache config files are located (in a running container).
- You have documented your configuration in your report.

RESUME:

In this step we want to install and configure a static httpd server apache wrapped in a docker machine. Then we add a static HTML content to test our server.

1- Make a Directory

After cloning the repository of the lab on our computer, we make a directory `docker-images` where we will put all our images. Then for this step we create a second directory `apache-php-image` where we create our Dockerfile for static HTTP server. This Dockerfile has instructions to build an images from one existant on Docker Hub. [Docker Hub website]

N.B : we realize all these commands in our vagrant machine

2- Build our image

- First we choose a php image from the official website : [PHP Image], we choose the version 7.0 of image.
- Second we write these lines into the Dockerfile:
  - `FROM php:7.0-apache` specify the version of php's image to download
  - `COPY src /var/www/html/` specify that when we build image we copy all contents from `src/` (in directory `apache-php-image` on local machine) into the directory `/var/www/html/` (directory in the container).
- Then we build our php's image from the Dockerfile in `apache-php-image` with the command : `docker build -t res/apache_php .` and create a container with name `dany/apache_php` . We use the option `-t` to give a name at the container

3- Running the container

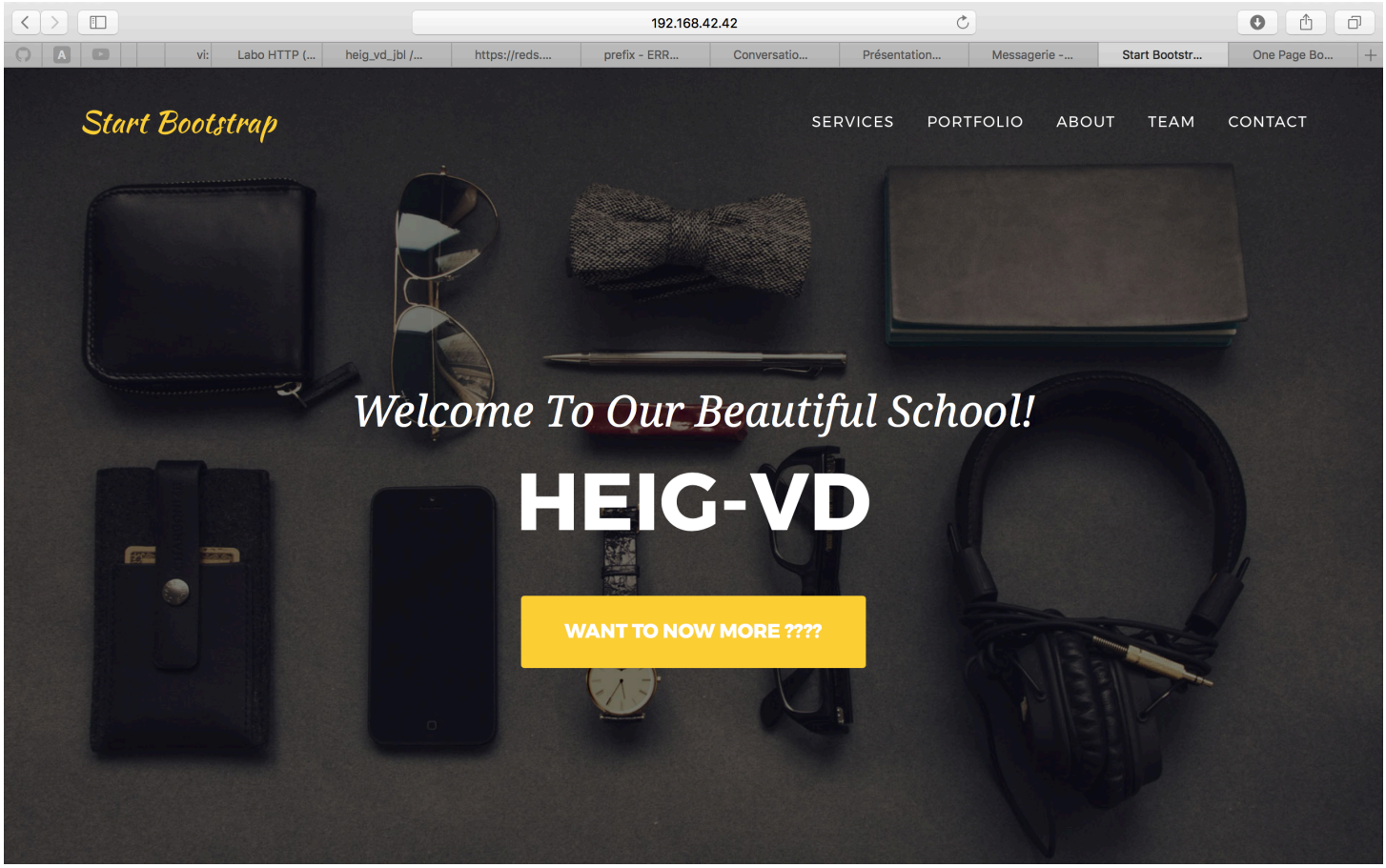
we run the container with the command : `docker run -d -p 9090:80 res/apache_php` , we use option `-d` to run the container in background , `-p 9090:80` to map the port's number of our vagrant machine with our container's port number that will allow to listen at this port (80 container) from 9090 in the local machine.

4- Realize the tests

In the container specifically in the directory `/var/www/html/` , we create a file `index.html` where we write some text into for testing communication (the server search that file at defaut entry by the client)to respond at a request from the local machine (in a terminal with command : telnet `192.168.42.42 9090` or in a browser with command: `http://192.168.42.42:9090/`) because if we don't do that the server will respond(*forbidden* you don't have permission to access on this server) with an error to indicate a missing file `.html` to show. To create a file we explore the container in running with command : `docker exec -it containerName /bin/bash` that command open a prompt in the container.
Tools: The containerName is obtained with command `docker ps` and the Ipaddress with the command : `docker inspect containerName | grep -i ipaddress` .

5- Amelioration of our HTML

We visit the website bootstrap to choose one example of page html with a nice look. After that we create a directory `apache-php-image/content` (we can use directly `src/`)to first save our `index.html` (we can modify index.html to show our text customize) and `our page html from startbootstrap` in our local machine because when a container is killed we lose all the configuration and file created. After the modification we kill the container with command `docker kill containerName or id` , we rebuild the image and rerun the container to apply the modification.
Capture:

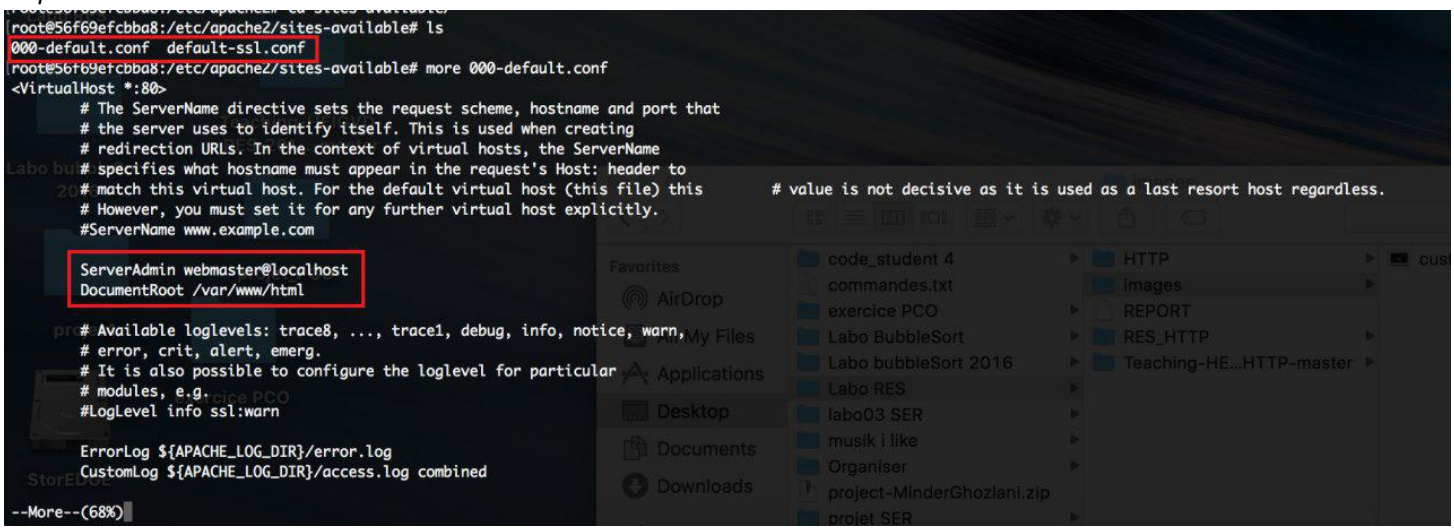


The section DocumentRoot specify the path where to find the file HTML when a request is done by the client on the port `80` , this path can be change.

6- Consult the apache configuration files

we can consult the apache configuration files in the `/etc/apache2/`, there we find sub-directory and files for each fonctions of configuration. Specifically in the directory `/etc/apache2/sites-available` we find two default files `000-default.conf` and `default-ssl.conf`

Capture:



Step 2: Dynamic HTTP server with express.js

Acceptance criteria

- You have a GitHub repo with everything needed to build the Docker image.
- You do a demo, where you build the image, run a container and access content from a browser.
- You generate dynamic, random content and return a JSON payload to the client.
- You cannot return the same content as the webcast (you cannot return a list of people).
- You don't have to use express.js; if you want, you can use another JavaScript web framework or event another language.
- You have documented your configuration in your report.

RESUME:

In this step, we want to write a dynamic application in Node.js which returns a JSON payload on GET requests to the client for example a browser, postman and telnet.

1- Make a Directory and create a Dockerfile

First we create a directory `docker-images/express-image` where we create our Dockerfile for this step and where we put all the config files (`src/`). To write in our Dockerfile, we go on `dockerHub` to find an official Node image to select a version of image to download.
Content of Dockerfile:

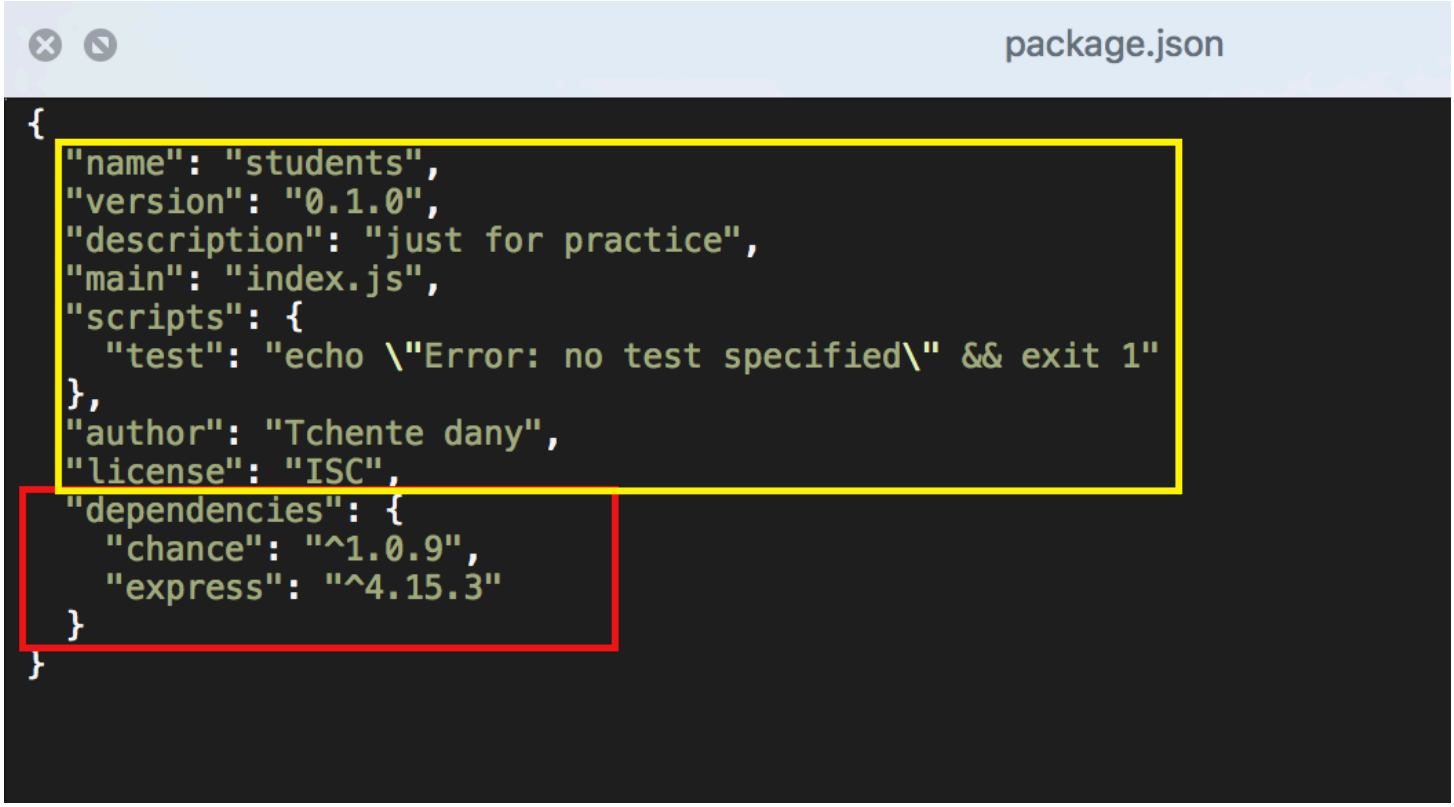
```
FROM node:6.10
MAINTAINER Dany Tchente
COPY src /opt/app
CMD ["node", "/opt/app/index.js"]
```

- `FROM node:6.10` specify the version of php's image to download

- `COPY src /opt/app` copy the content in src into /opt/app in the container
- `CMD ["node", "/opt/app/index.js"]` specify the command to execute when we run a container. We can also use ENTRYPOINT as CMD.

2- Create a Docker image

First we create a directory `docker-images/express-image/src` where we store node-modules and other file use by the container, then we execute the command `npm init` because we want to work with node.js. After that we obtain a package.json with zero dependency, to add dependency we install the modul chance.js which generate random value with the command : `npm install --save chance` the option `--save` specify that we want to save dependencies from module chance. Secondly we create our `index.js` file which can be use by the server to generate random identity. After that we edit `index.js` with our implementation to generate some value for example the random location, city and country. After editing the file we can execute it with command : `node index.js` . The dependencies are saved in the directory `node_modules` *Capture:*



In the Red Box we see the dependencies added an in yellow box our data.

3- Build a Docker image and Run a container Node

After creation of application `index.js` , we build a docker image and run the container with the command : `cd /vagrant/RES/Teaching-HEIGVD-RES-2016-Labo-HTTPInfra/docker-images/express-image` `docker build -t res/node .` `docker run res/node`

4- Build a Docker image and Run a container Express

After testing our node docker to generate random locations, we install `express.js` with command : `npm install --save express` which add dependencies in `package.json` . In this case we use the framework express to communicate with the server with a request (`GET /` we can for example use `GET /location`) for example our function generate a random list of location and listen on the port `3000` when he receive a request `GET /` . Then we

modify our application `index.js` like that : *Capture:*

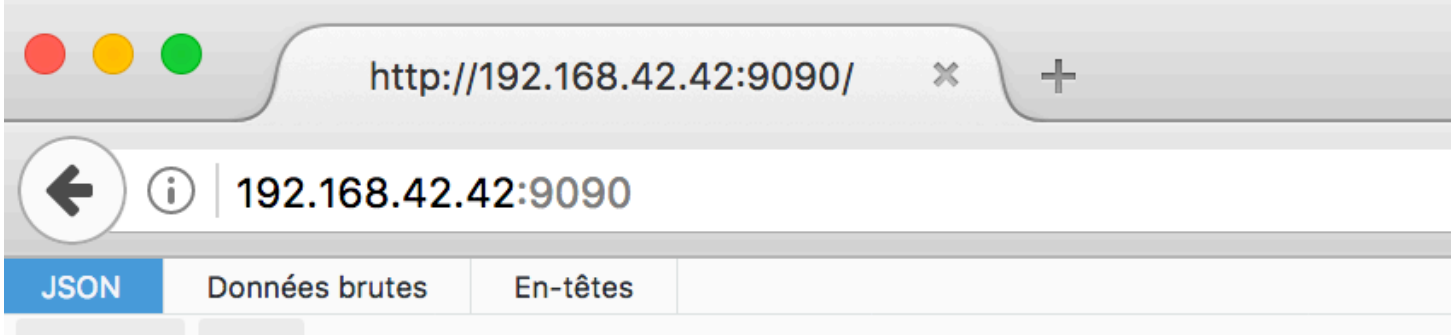
```
var Chance = require('chance');
var chance = new Chance();

var express = require('express');
var app = express();


app.get('/', function (req, res) {
    res.send(generateLocation());
});

app.listen(3000, function () {
    console.log('Accepting HTTP requests on port 3000!');
});

function generateLocation(){
    var numberOfLocation = chance.integer({
        min: 5,
        max: 20
    });

    console.log(numberOfLocation);
    var locations = [];

    for(var i = 0; i < numberOfLocation; i++){

        var province = chance.province({full: true});

        var country = chance.country({country: 'fr', full: true}); // to return a full name

        var city = chance.city();

        var postal = chance.postal();

        var street = chance.street();


        locations.push({
            province: province,
            country: country,
            city: city,
            postal: postal,
            street : street
        });
    };
    console.log(locations);
    return locations;
}
```

we obtains this result after apply these commands :

`cd /vagrant/RES/Teaching-HEIGVD-RES-2016-Labo-HTTPInfra/docker-images/express-image`

`docker build -t res/express_locations .` `docker run -d -p 9090:3000 res/express_locations`

Capture:



Enregistrer Copier


```
0:
    province:   "Alberta"
    country:    "Luxembourg"
    city:       "Vaipvo"
    postal:     "N2E 4J3"
    street:     "Sidil Mill"
1:
    province:   "Northwest Territories"
    country:    "Mongolia"
    city:       "Firpabe"
    postal:     "X7J 7B0"
    street:     "Zephi Parkway"
2:
    province:   "Northwest Territories"
    country:    "Namibia"
    city:       "Doshabe"
    postal:     "C9F 0M3"
    street:     "Noszo River"
3:
    province:   "British Columbia"
    country:    "Guadeloupe"
    city:       "Rowikfo"
    postal:     "P3H 7I3"
    street:     "Gesi Loop"
4:
    province:   "Yukon"
    country:    "Slovakia"
    city:       "Lulematev"
    postal:     "K3C 2I3"
    street:     "Wecte Park"
5:
    province:   "Nunavut"
    country:    "Armenia"
    city:       "Asninad"
    postal:     "N0J 0S1"
    street:     "Fessa Heights"
6:
    province:   "Prince Edward Island"
```

```
country:    "Senegal"
city:       "Ufufilo"
postal:     "H1L 109"
street:     "Upogos Heights"
```

Step 3: Reverse proxy with apache (static configuration)

Acceptance criteria

- You have a GitHub repo with everything needed to build the Docker image for the container.
- You do a demo, where you start from an "empty" Docker environment (no container running) and where you start 3 containers: static server, dynamic server and reverse proxy; in the demo, you prove that the routing is done correctly by the reverse proxy.
- You can explain and prove that the static and dynamic servers cannot be reached directly (reverse proxy is a single entry point in the infra).
- You are able to explain why the static configuration is fragile and needs to be improved.
- You have documented your configuration in your report.

RESUME:

In this step we will install a reverse proxy empty (no content) to communicate with our two server `dynamic server` and `static server` to improve security and respect the same origin policy (specify that an ajax's request can always communicate with a server in the same domain). For this step we need to configure the container for this task.

1- Run the container static and dynamic

First we run the containers with commands : `docker run -d --name apache_static dany/apache` `docker run -d --name apache_dynamic dany/express_locations` the option `--name` specify the name of our container.

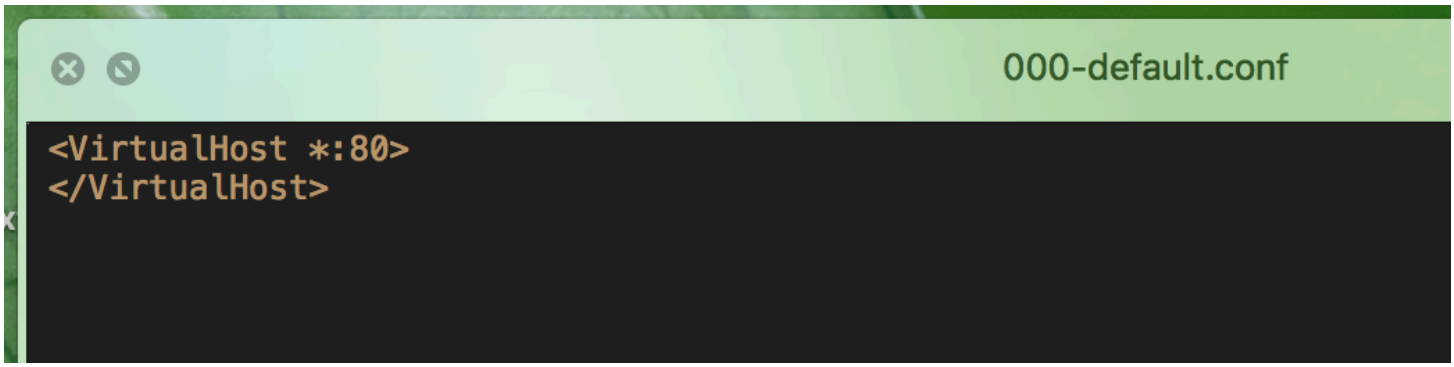
2- Install Reverse-proxy

First we get the ipaddress of our containers because we need to map them with our reverse-proxy. After that we create directory `docker-images/apache-reverse-proxy` then we create our Dockerfile like that:

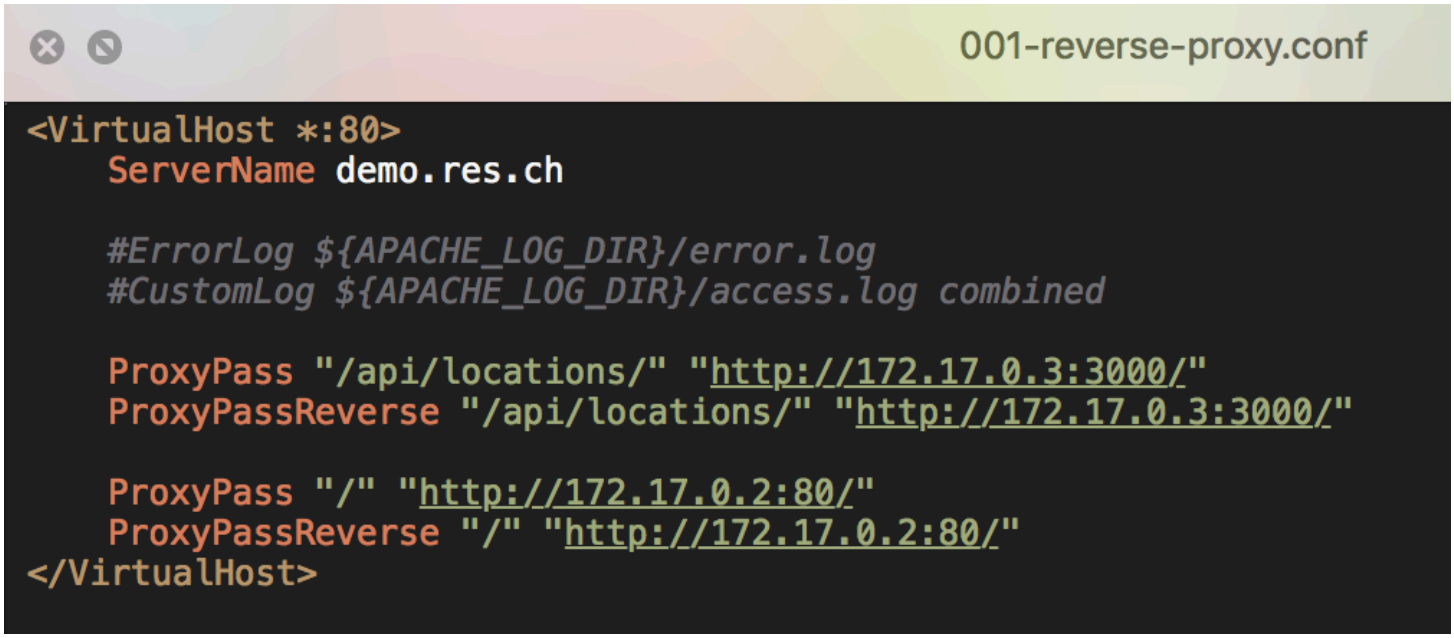
```
FROM php:7.0-apache
MAINTAINER dany Simo
RUN apt-get update && \
apt-get install -y vim
COPY conf/ /etc/apache2
RUN a2enmod proxy proxy_http
RUN a2ensite 000-* 001-*
```

- `RUN apt-get update && \ apt-get install -y vim` we install the editor vim when we run our container
- `COPY conf/ /etc/apache2` we copy the content of conf in the local machine into the apache2 configuration
- `RUN a2enmod proxy proxy_http` we enable module `proxy` and `proxy_http`
- `RUN a2ensite 000-* 001-*` we enable our configuration like on the captcha

Before that we connect to the container (command : `docker run -it php:7.0-apache`) php:7.0-apache to see where we can apply the configuration.There we copy the `000-default.conf` to `001-reverse-proxy.conf` then we modify it like that: *Capture:*



Capture:



N.B: ProxyPass and ProxyPassReverse allows us to access to our container with dynamic server via `/api/locations` and static server via `/`.

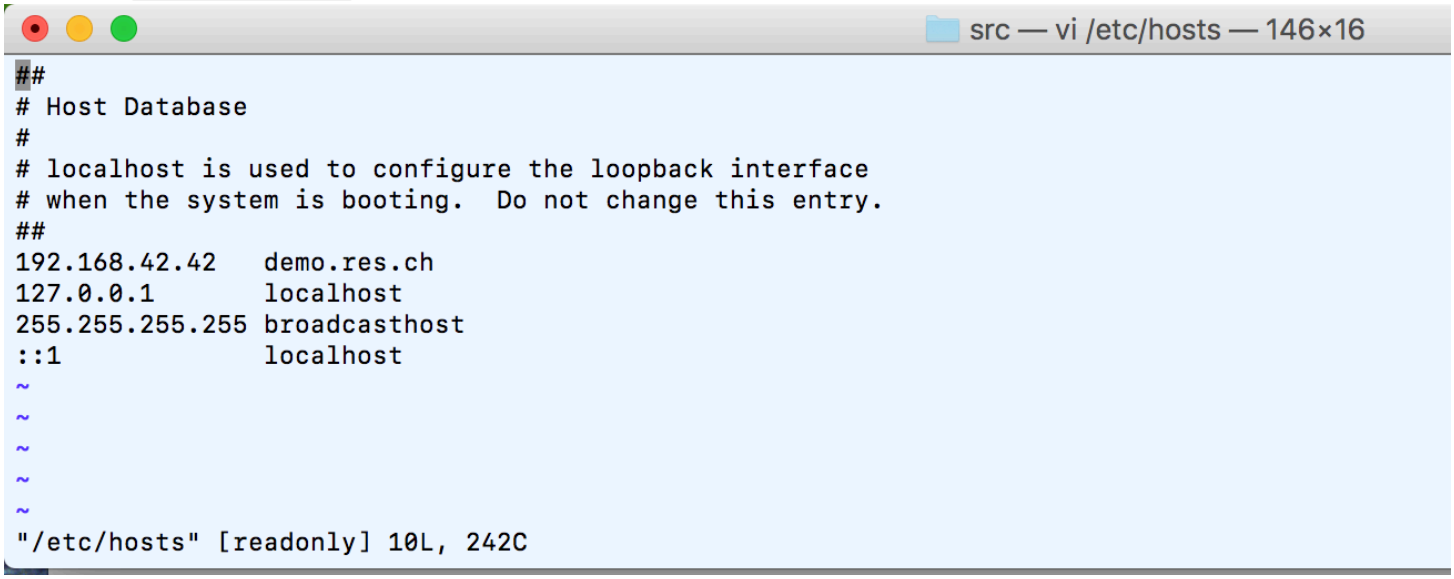
Then we reload the service apache with command : `service apache reload` or `service apache restart`. if we find an error (ProxyPass Invalid command) we need to enable proxy and proxy_http with command `a2enmod proxy` and `a2enmod proxy_http` then we reload. After add our configuration files and our Dockerfile we build and run our container with command:
`cd /vagrant/RES/Teaching-HEIGVD-RES-2016-Labo-HTTPInfra/docker-images/apache-reverse-proxy`
`docker build -t dany/apache_rp .` `docker run -p 8080:80 dany/apache_rp` Then in a terminal we execute these commands :

```
telnet 192.168.42.42 8080
GET / HTTP/1.0
Host: demo.res.ch
```

for access to static server and we need to precise the host. This host can be change if we want and the port number `8080`.

```
telnet 192.168.42.42 8080
GET /api/locations/ HTTP/1.0
Host: demo.res.ch
```

for access to dynamic server. Then in a browser we receive an error `forbidden` when we want to access via `192.168.42.42:8080` to resolve it: we need to modify the file hosts database in our local machine and add our vagrant's ipaddress `192.168.42.42` like on the captcha : *Capture:*

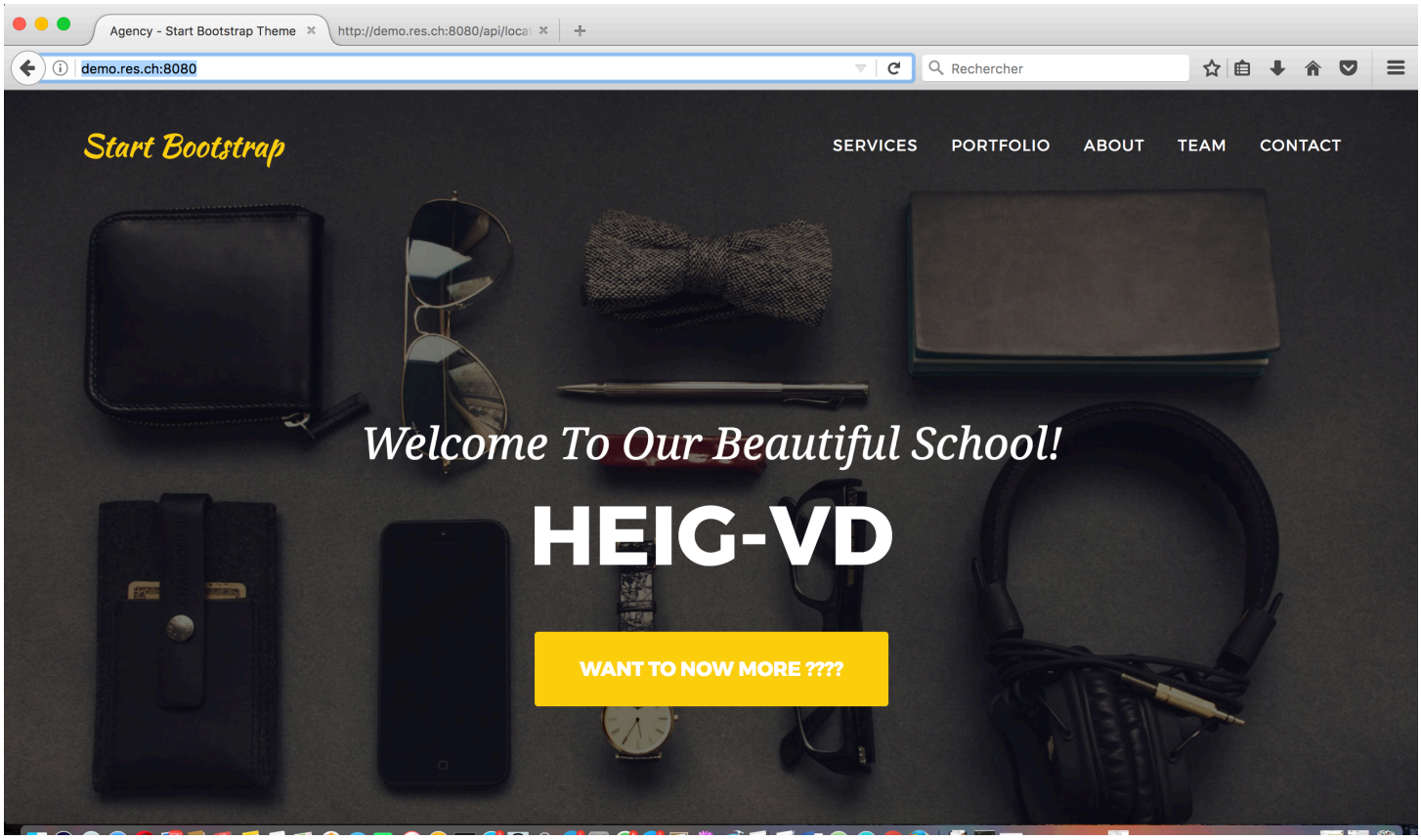


Then we can access via commands :

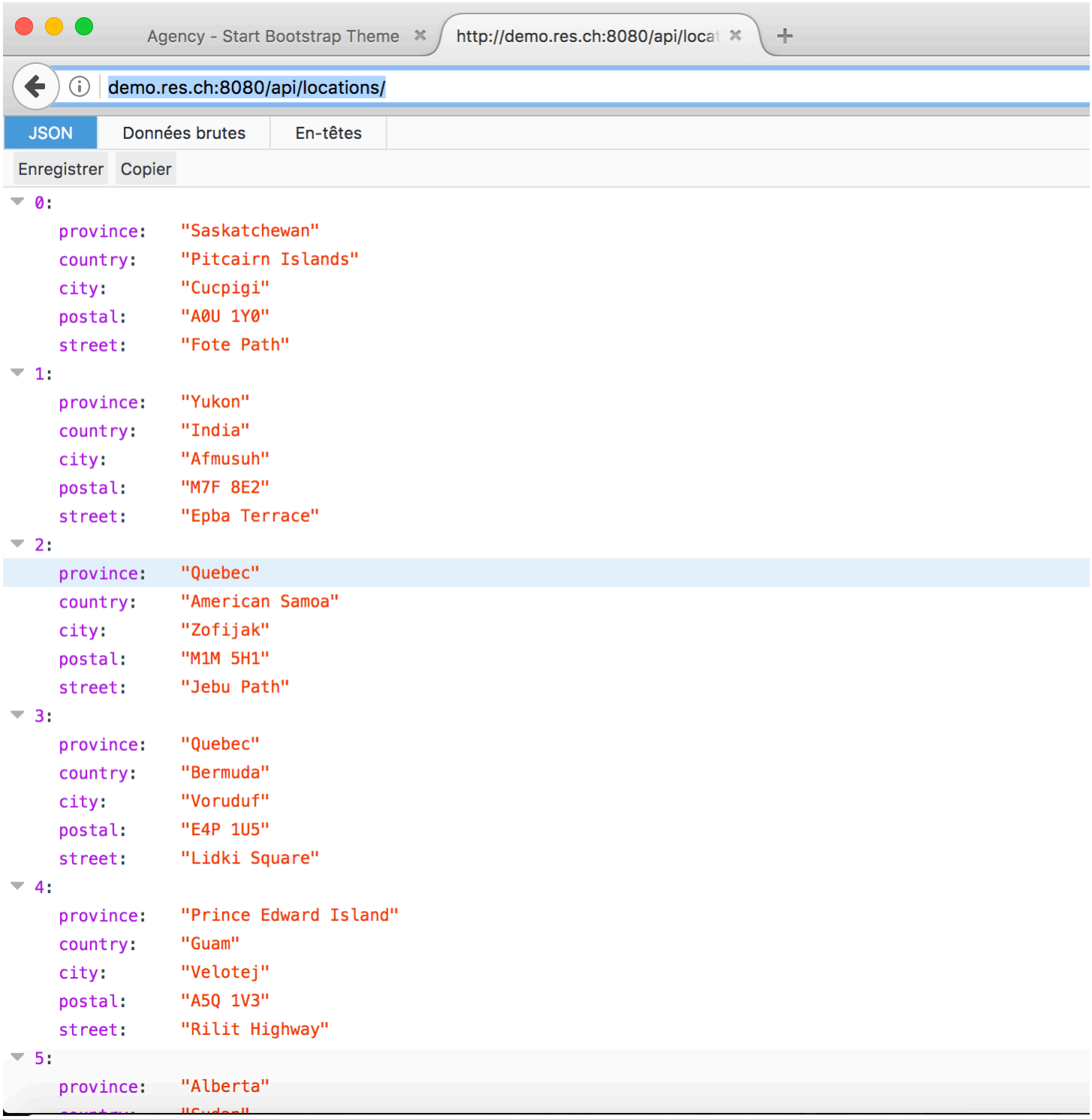
```
http://demo.res.ch:8080/
http://demo.res.ch:8080/api/locations/
```

Results on the capture:

Capture:



Capture:



Step 4: AJAX requests with JQuery

Acceptance criteria

- You have a GitHub repo with everything needed to build the various images.
- You do a complete, end-to-end demonstration: the web page is dynamically updated every few seconds (with the data coming from the dynamic backend).
- You are able to prove that AJAX requests are sent by the browser and you can show the content of th responses.
- You are able to explain why your demo would not work without a reverse proxy (because of a security restriction).
- You have documented your configuration in your report.

RESUME:

In this step we need to use the library (javaSript) JQuery to make an ajax request.

1- Stop and Restart our containers

First we need to stop our containers with th command :

```
docker kill `docker ps -qa`
docker rm `docker ps -qa`
```

And then restart our containers

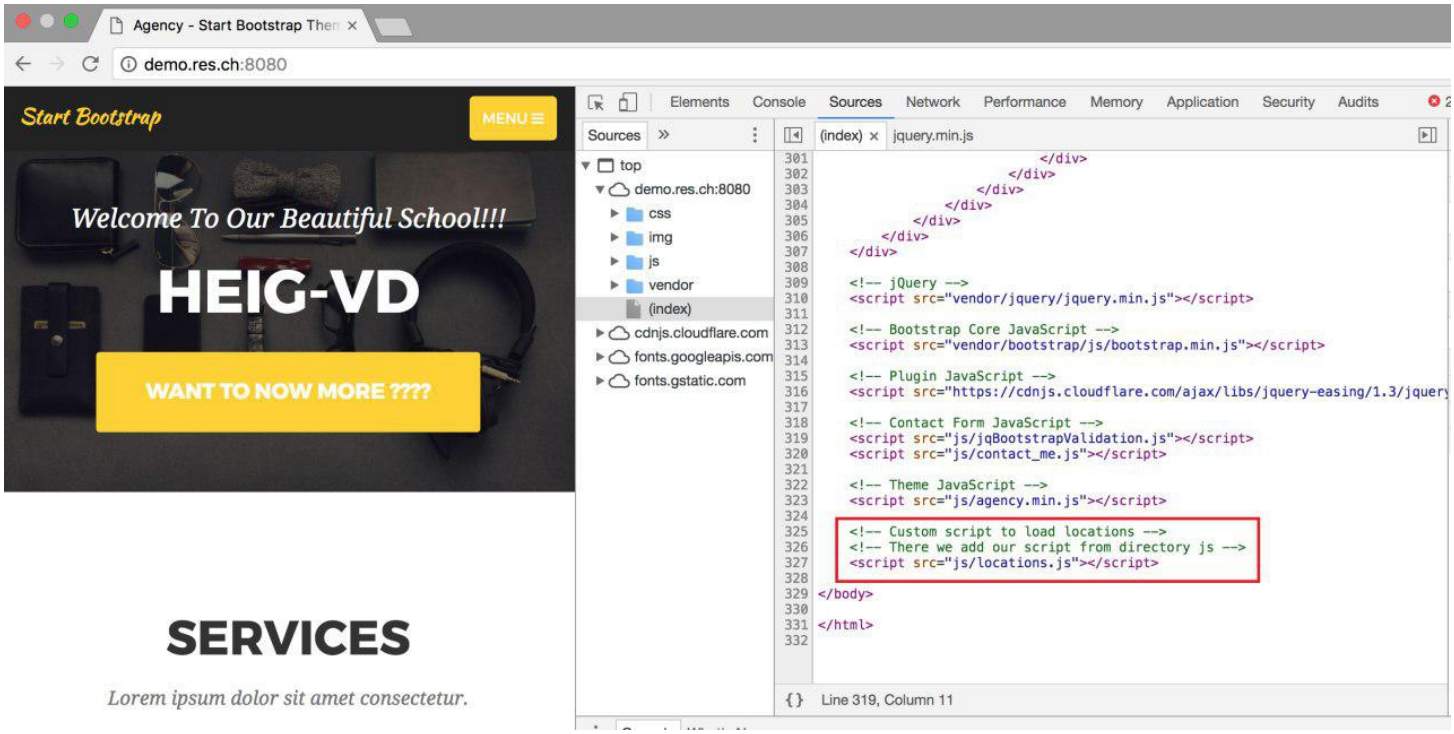
```
docker run -d --name apache_static res/apache_php
docker run -d --name express_dynamic res/express_locations
docker run -d -p 8080:80 --name apache_rp res/apache_rp
```

2- Log into static HTTP container

we use the command `docker exec -it containerName /bin/bash` to connect to the container and then apply our modification on the copy of our index.html

3- Add the script in the index.html

At the bottom of our page we can see the library JQuery and all the script using on the page. There we add our script like on the capture. *Capture:*



4- Create our javascript locations

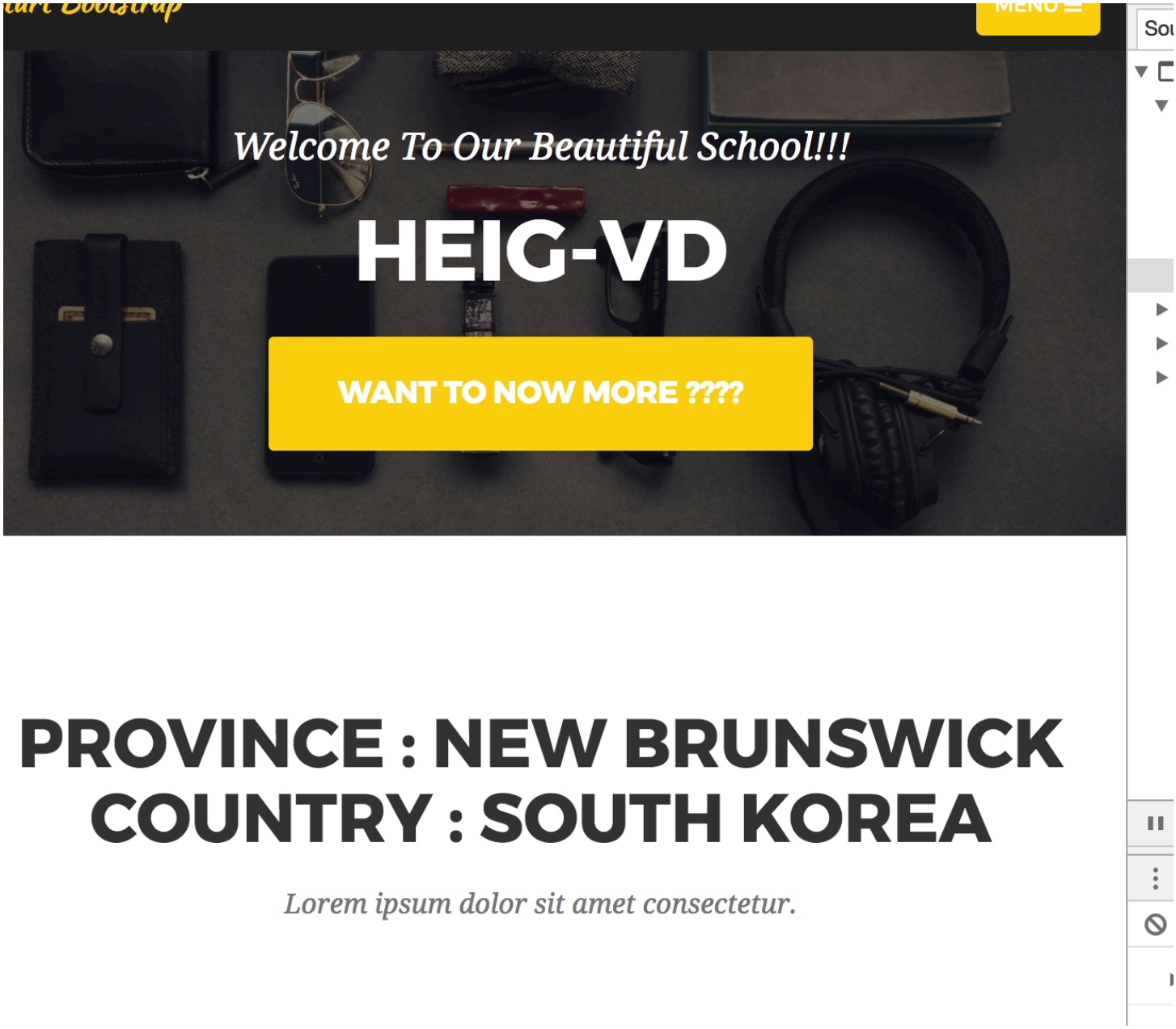
We create a javascript file to modify dynamically our static page html with our list of locations. *Capture:*

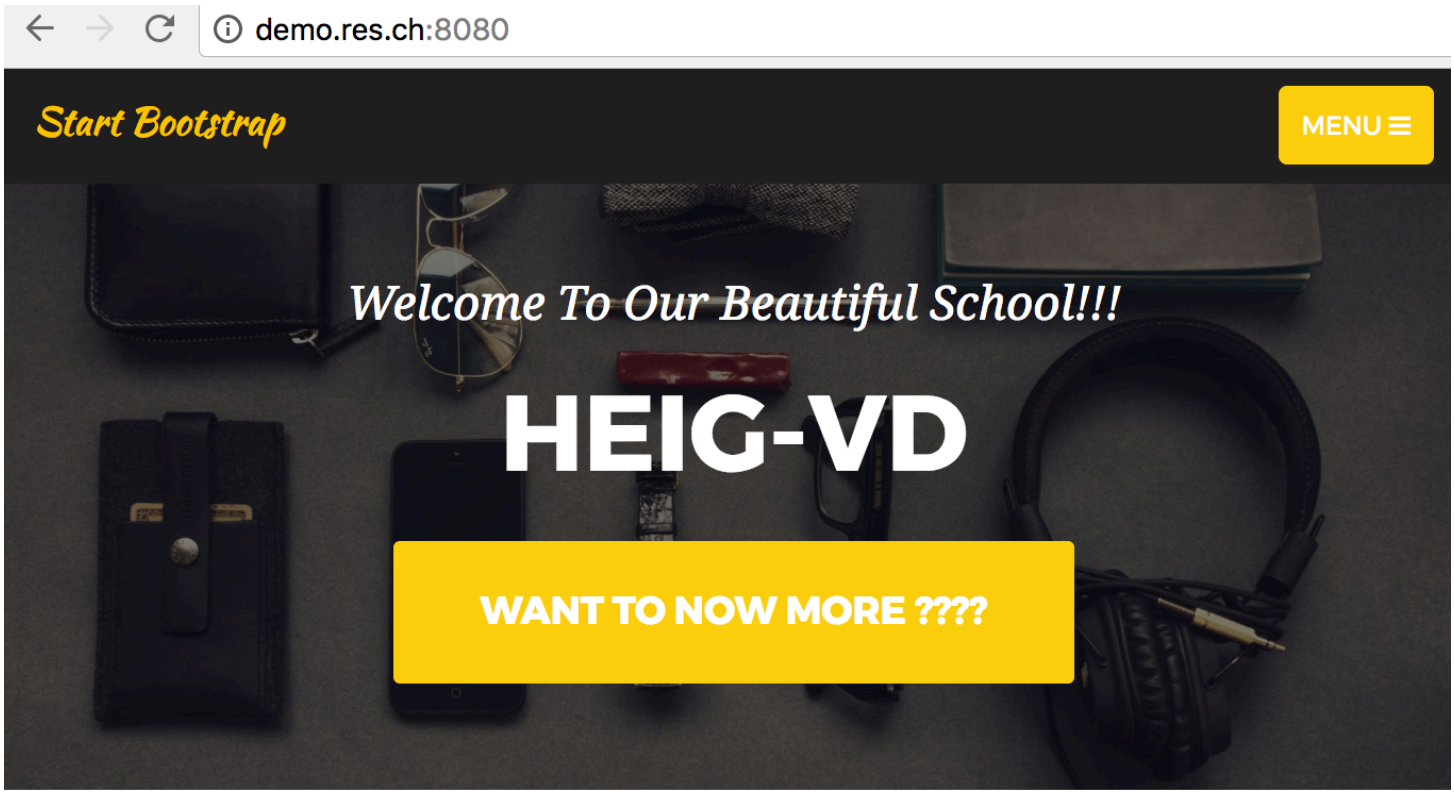
locations.js — js (git: fb-ajax-jquery)

```
$(function(){


        function loadLocations(){
                $.getJSON("/api/locations/", function(locations){
                        console.log(locations);
                        var message = "No Locations to Show";
                        if(locations.length > 0){
                                message = "Province : " + locations[0].province + "\n" + "Country : " + locations[0].country;
                        }
                        $(".section-heading").text(message);
                });
        };

        loadLocations();
        setInterval(loadLocations, 3000);

});
```

5- Test

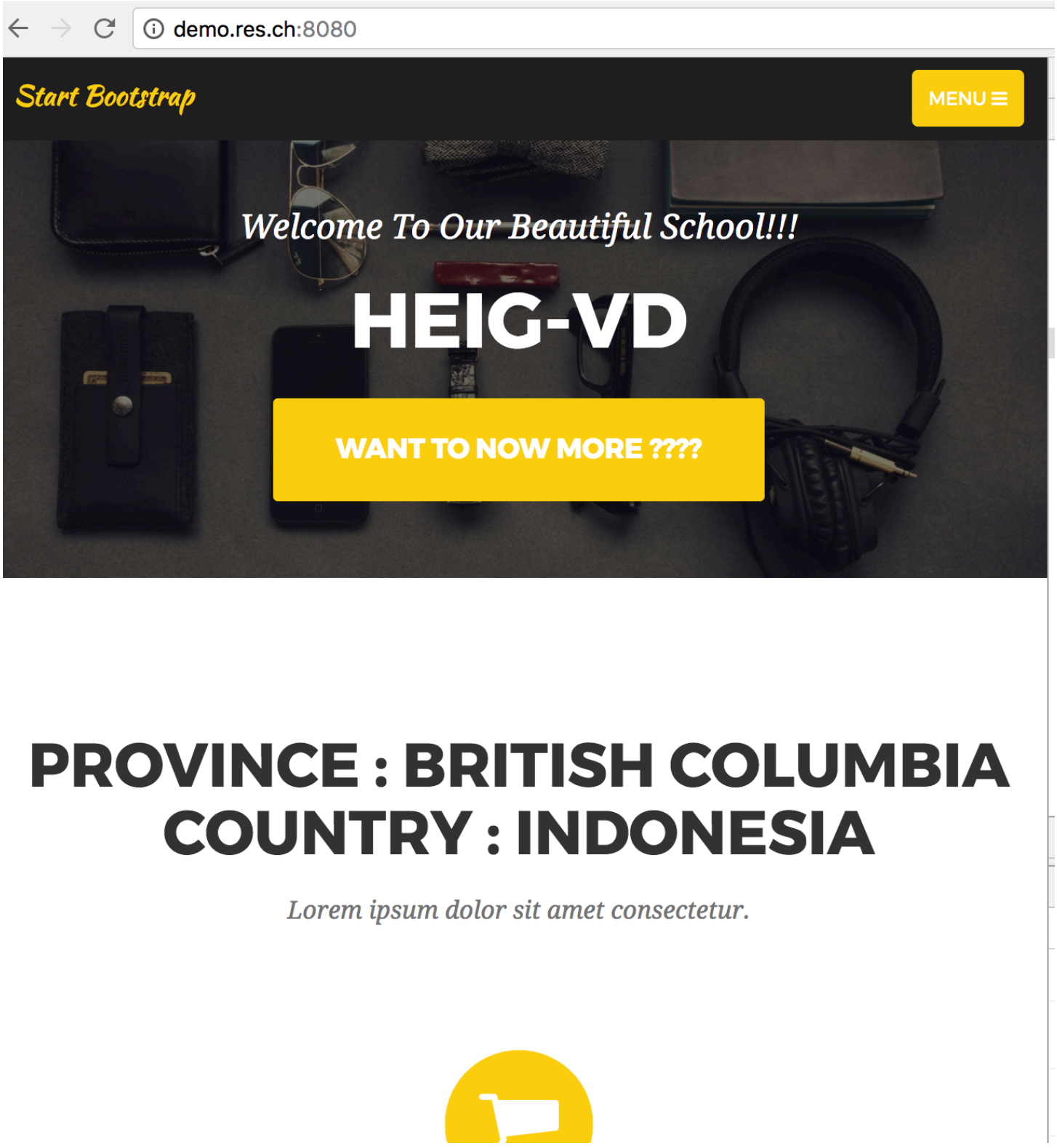
Capture:





PROVINCE : NEW BRUNSWICK
COUNTRY : AUSTRALIA

Lorem ipsum dolor sit amet consectetur.



Step 5: Dynamic reverse proxy configuration

Acceptance criteria

- You have a GitHub repo with everything needed to build the various images.
- You have found a way to replace the static configuration of the reverse proxy (hard-coded IP adresses) with a dynamic configuration.
- You may use the approach presented in the webcast (environment variables and PHP script executed when the reverse

proxy container is started), or you may use another approach. The requirement is that you should not have to rebuild the reverse proxy Docker image when the IP addresses of the servers change.
- You are able to do an end-to-end demo with a well-prepared scenario. Make sure that you can demonstrate that everything works fine when the IP addresses change!
- You are able to explain how you have implemented the solution and walk us through the configuration and the code.
- You have documented your configuration in your report.

RESUME:

In this Step we need to replace our hard-coded IP addresses for the container in reverse proxy configuration. To do that we need to communicate with the containers from our local machine and then set static Ipaddresses in environment variable (local machine) when starting a Docker Container. We will add a script in the reverse-proxy container which will set Ipaddresses of container from local machine by generating a configuration file. Then this action replace the static configuration with IPaddresses to dynamic configuration. We will use also a configuration which generate php's template to generate our configuration file from our script (to modify IpAddresses).

1- Set configuration with Environment variables and Invoke a script

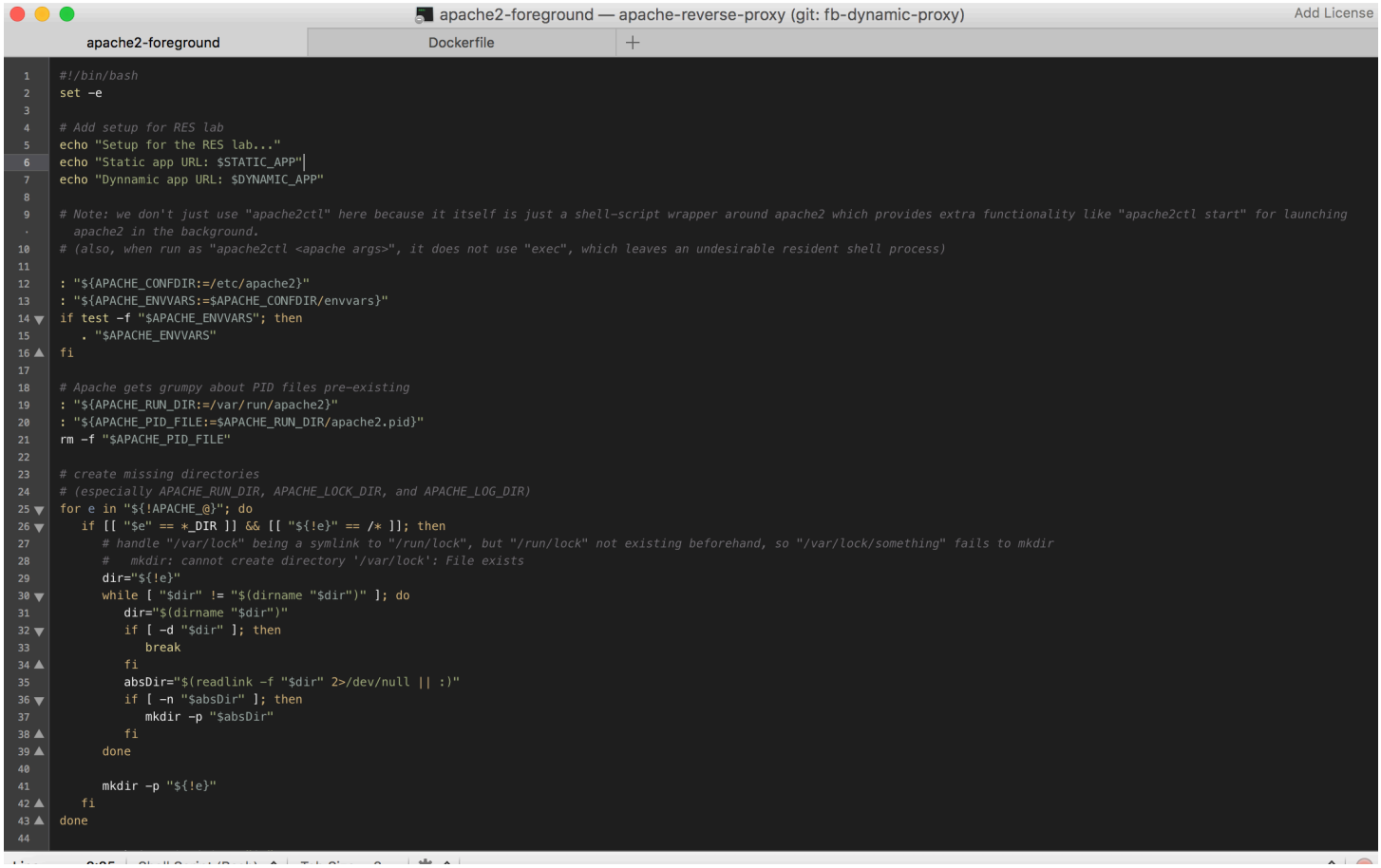
First we go in library php in github apache2-foreground then we see how he configure the start of server apache. We create file `apache2-foreground` as the same on github repository and we modify our Docker file :
Capture:

Dockerfile — apache-reverse-proxy (git: fb-dynamic-proxy)

apache2-foreground | Dockerfile

```
1  FROM php:7.0-apache
2
3  MAINTAINER dany Simo
4
5  RUN apt-get update && \
6     apt-get install -y vim
7
8  COPY conf/ /etc/apache2
9  COPY apache2-foreground /usr/local/bin/
10
11
12 RUN a2enmod proxy proxy_http
13 RUN a2ensite 000-* 001-*
14
15
```

There we copy our script (apache2-foreground) in `/usr/local/bin` in the reverse proxy container.
Capture:



There we add setup for RES lab and our environment variable `STATIC_APP` and `DYNAMIC_APP`. After that we build and run our container with commands `docker build -t dany/apache_rp .` and `docker run -e STATIC_APP=172.17.0.2:80 -e DYNAMIC_APP=172.17.0.3:3000 dany/apache_rp`, then we obtain:

Capture:

```
vagrant@RES:/vagrant/RES/Teaching-HEIGVD-RES-2017-Labo-HTTPInfra/docker-images/apache-reverse-proxy$ docker run -e STATIC_APP=172.17.0.2:80 -e DYNAMIC_APP=172.17.0.3:3000 dany/apache_rp
Setup for the RES lab...
Static app URL: 172.17.0.2:80
Dynnamic app URL: 172.17.0.3:3000
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.17.0.2. Set the 'ServerName' directive globally to suppress this message
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.17.0.2. Set the 'ServerName' directive globally to suppress this message
[Thu Jun 08 08:54:19.512196 2017] [mpm_prefork:notice] [pid 1] AH00163: Apache/2.4.10 (Debian) PHP/7.0.19 configured -- resuming normal operations
[Thu Jun 08 08:54:19.512288 2017] [core:notice] [pid 1] AH00094: Command line: 'apache2 -D FOREGROUND'
```

We can see that the script are invoke in the Dockerfile from apache2 when it starts.

2- Use PHP to create a template for the RP configuration file

PHP allows to write applications web, but here we will use php as a script langage to inject environment variables in the configuration file. we create a directory `template` where we put our `config-template.php` and then add our config from container reverse-proxy `001-reverse-proxy.conf`. We use the method from php `getenv('ENV')` to get the environment variables. *Capture:*

config-template.php — template (git: fb-dynamic-proxy)

```
<?php
$static_app = getenv('STATIC_APP');
$dynamic_app = getenv('DYNAMIC_APP');
?>

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

   ProxyPass '/api/locations/' 'http://<?php print "$dynamic_app"?>/'
   ProxyPassReverse '/api/locations/' 'http://<?php print "$dynamic_app"?>/'

   ProxyPass '/' 'http://<?php print "$static_app"?>/'
   ProxyPassReverse '/' 'http://<?php print "$static_app"?>/'
</VirtualHost>
```

Then we set the `apache2-foreground` to copy the result of the script to the `001-reverse-proxy.conf` file.

Capture:

apache2-foreground

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

# Add setup for RES lab
echo "Setup for the RES lab..."
echo "Static app URL: $STATIC_APP"
echo "Dynnamic app URL: $DYNAMIC_APP"

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

# Note: we don't just use "apache2ctl" here because it itself is just a shell-script wrapper around apache2 which provides extra fun
# (also, when run as "apache2ctl <apache args>", it does not use "exec", which leaves an undesirable resident shell process)

: "${APACHE_CONFDIR:=/etc/apache2}"
: "${APACHE_ENVVARS:=$APACHE_CONFDIR/envvars}"
if test -f "$APACHE_ENVVARS"; then
    . "$APACHE_ENVVARS"
fi

# Apache gets grumpy about PID files pre-existing
: "${APACHE_RUN_DIR:=/var/run/apache2}"
: "${APACHE_PID_FILE:=$APACHE_RUN_DIR/apache2.pid}"
rm -f "$APACHE_PID_FILE"

# create missing directories
# (especially APACHE_RUN_DIR, APACHE_LOCK_DIR, and APACHE_LOG_DIR)
for e in "${!APACHE_@}"; do
    if [[ "$e" == *_DIR ]] && [[ "${!e}" == /* ]]; then
        # handle "/var/lock" being a symlink to "/run/lock", but "/run/lock" not existing beforehand, so "/var/lock/something" fails
        #   mkdir: cannot create directory '/var/lock': File exists
        dir="${!e}"
        while [ "$dir" != "$(dirname "$dir")" ]; do
            dir="$(dirname "$dir")"
            if [ -d "$dir" ]; then
                break
            fi
            absDir="$(readlink -f "$dir" 2>/dev/null || :)"
            if [ -n "$absDir" ]; then
                mkdir -p "$absDir"
            fi
        done

        mkdir -p "${!e}"
    fi
done

exec apache2 -DFOREGROUND "$@"
```

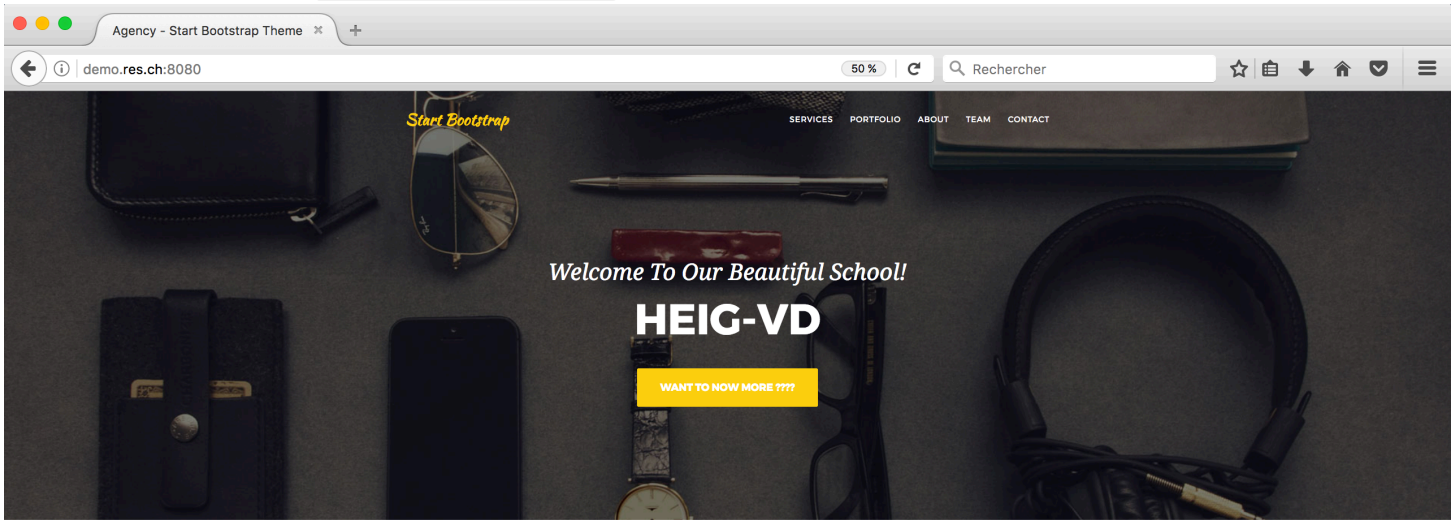
The modification is the white line in the capture.

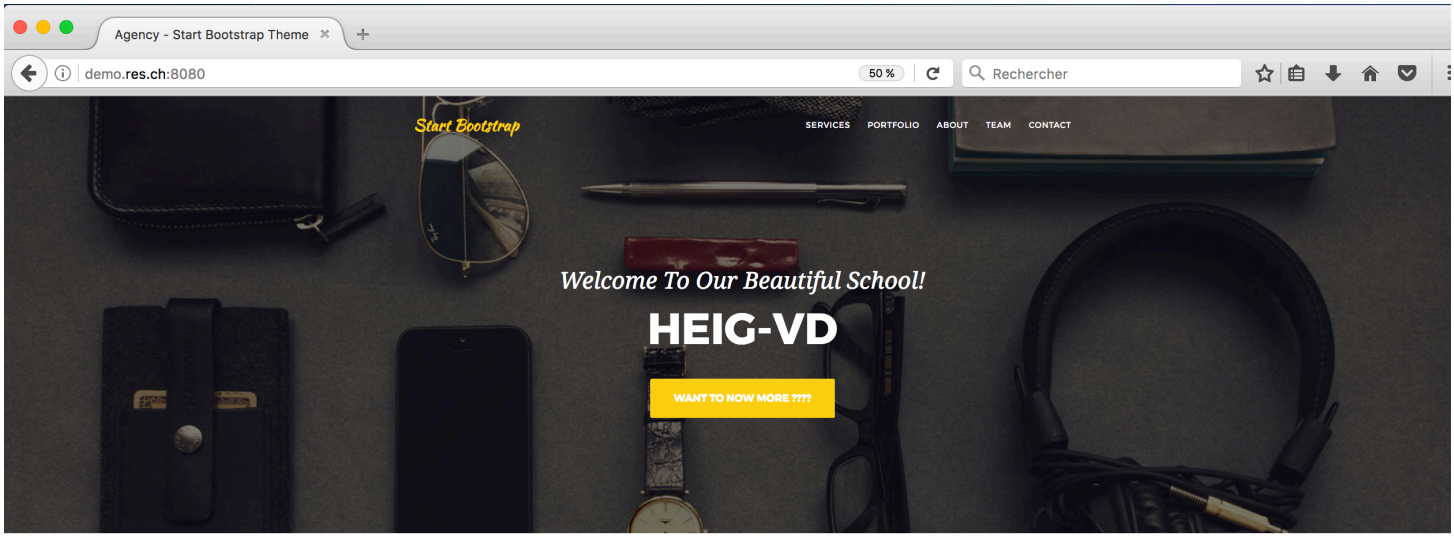
3- Tests

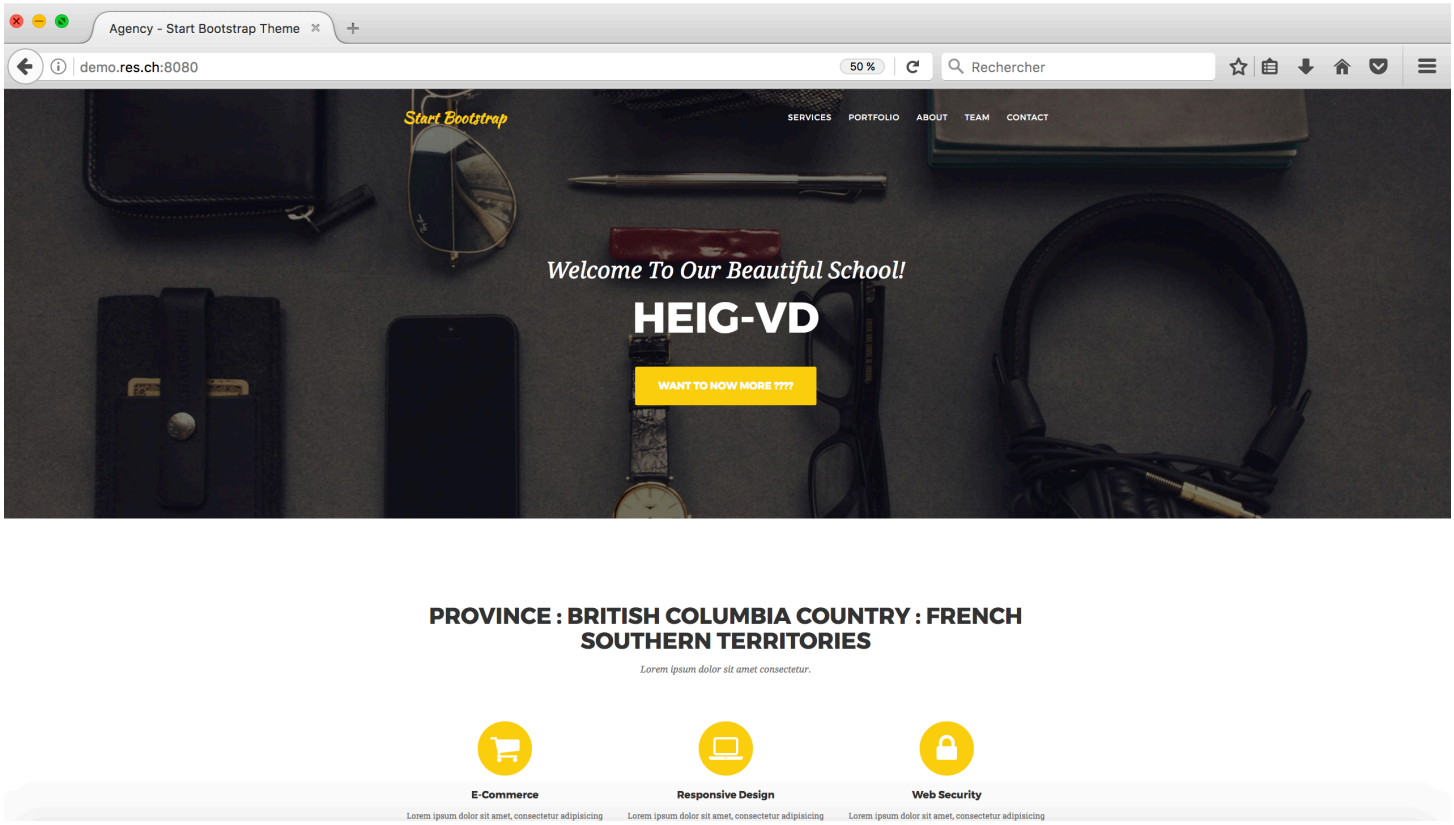
After apply command

```
docker run -e STATIC_APP=172.17.0.5:80 -e DYNAMIC_APP=172.17.0.8:3000 -p 8080:80 dany/apache_rp
```

The Ipaddress obtain with `docker inspect ...` and run the docker container we obtain: *Capture:*







Additional steps to get extra points on top of the "base" grade

Load balancing: multiple server nodes (0.5pt)

- You extend the reverse proxy configuration to support **load balancing**.
- You show that you can have **multiple static server nodes** and **multiple dynamic server nodes**.
- You prove that the **load balancer** can distribute HTTP requests between these nodes.
- You have documented your configuration and your validation procedure in your report.

Load balancing: round-robin vs sticky sessions (0.5 pt)

- You do a setup to demonstrate the notion of sticky session.
- You prove that your load balancer can distribute HTTP requests in a round-robin fashion to the dynamic server nodes (because there is no state).
- You prove that your load balancer can handle sticky sessions when forwarding HTTP requests to the static server nodes.
- You have documented your configuration and your validation procedure in your report.

Dynamic cluster management (0.5 pt)

- You develop a solution, where the server nodes (static and dynamic) can appear or disappear at any time.
- You show that the load balancer is dynamically updated to reflect the state of the cluster.
- You describe your approach (are you implementing a discovery protocol based on UDP multicast? are you using a tool such as serf?)
- You have documented your configuration and your validation procedure in your report.

Management UI (0.5 pt)

- You develop a web app (e.g. with express.js) that administrators can use to monitor and update your web infrastructure.
- You find a way to control your Docker environment (list containers, start/stop containers, etc.) from the web app. For instance, you use the Dockerode npm module (or another Docker client library, in any of the supported languages).
- You have documented your configuration and your validation procedure in your report.