

# HTTP Infrastructure

Rapport du laboratoire de RES

Tano Iannetta

Loan Lassalle

15.06.2017



## Table des matières

Table des matières .....	3
1    Serveur HTTP statique avec apache httpd .....	4
1.1    Fichiers de configuration d'un serveur web Apache .....	5
2    Serveur HTTP dynamique avec express.js .....	6
3    Proxy inverse avec apache (configuration statique) .....	10
3.1    Pourquoi les serveurs ne peuvent pas être atteints directement ? .....	12
3.2    Pourquoi la configuration statique est-elle fragile et doit être améliorée ? .....	12
4    Requêtes AJAX avec JQuery .....	13
4.1    Est-ce que les demandes AJAX sont envoyées par le navigateur ? .....	16
4.2    Pourquoi la démonstration ne fonctionnerait pas sans un proxy inverse ? .....	16
5    Configuration dynamique du proxy inversé .....	17
6    Etapes additionnelles .....	19
6.1    Répartition de charge .....	19
6.2    Répartition de charge avec affinité de serveur .....	23
6.3    Gestion dynamique des agrégats de conteneurs .....	24
6.4    Interface d'administration .....	24
7    Procédure de validation .....	24
8    Commandes diverses .....	29
8.1    Supprime tous les conteneurs .....	29
8.2    Supprime toutes les images .....	29
8.3    Accède au terminal de la machine virtuel Docker .....	29

## 1 Serveur HTTP statique avec apache httpd

### [Repository GitHub Task 1](#)

La première étape a été de créer une image Docker. Pour cela, nous avons utilisé l'image officielle **PHP**. Nous avons décidé d'utiliser cette image en raison de sa simplicité d'utilisation, la mise à disposition de fonctionnalités complètes et du format compact du fichier Dockerfile.

```
FROM php:7.0-apache

LABEL maintainer Tano Iannetta <tano.iannetta@heig-vd.ch>
LABEL maintainer Loan Lassalle <loan.lassalle@heig-vd.ch>

COPY ./content/ /var/www/html
```

Pour rendre la démonstration plus intéressante, nous avons décidé de récupérer un modèle de site Internet à [cette adresse](#). Une fois le modèle récupéré, nous avons placé les fichiers sources dans le dossier **content**.

Grâce à la définition du fichier Dockerfile, le contenu du répertoire **content** est copié vers le répertoire **/var/www/html** de la machine cible afin de fournir au serveur web Apache un site Internet de base. Ainsi, nous avons pu tester le bon fonctionnement du serveur web Apache avec un site Internet basique mais de qualité.

Une fois le fichier Dockerfile et le site Internet prêt, nous avons construit l'image.

```
docker build -t res/apache_php ./docker-images/apache-php-image
```

Après la construction de l'image, nous avons pu exécuter un conteneur en se basant sur la précédente image.

Pour pouvoir accéder au serveur Apache depuis un navigateur Internet de la machine hôte, il est nécessaire de faire du « port mapping ». Ceci permet de faire correspondre un port du conteneur à un port de la machine virtuelle Docker et ainsi d'accéder au conteneur à partir la machine hôte.

```
docker run -d --name apache_static -p 9090:80 res/apache_php
```

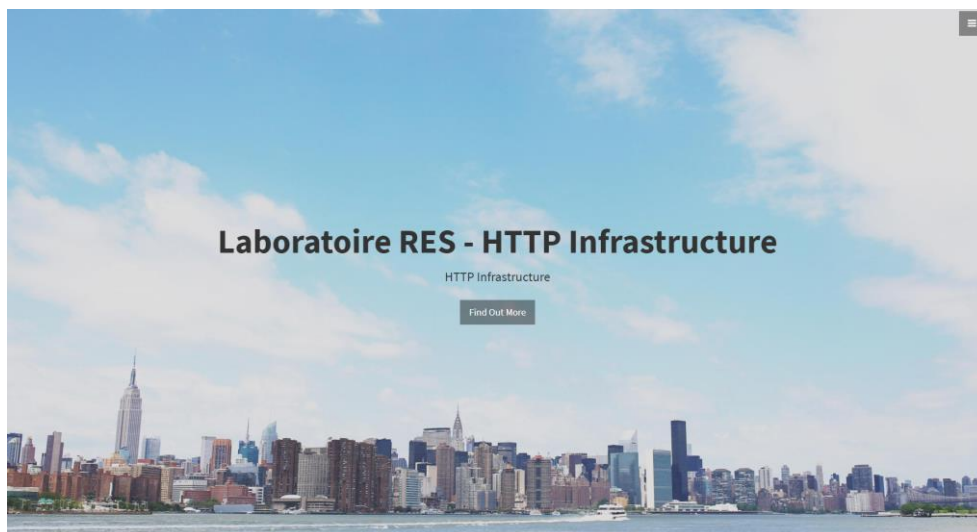


Figure 1 : Page d'accueil du serveur HTTP statique

A la fin de cette étape, nous avons réussi à mettre en place un serveur web Apache statique au sein de notre infrastructure HTTP.

### 1.1 Fichiers de configuration d'un serveur web Apache

Pour accéder au système de fichiers du conteneur nous avons utilisé la commande suivante :

```
docker exec -it <containerID> /bin/bash
```

La configuration d'un serveur web Apache est séparée en plusieurs parties. Ces différentes parties sont définies par les fichiers contenus dans le répertoire :

```
/etc/apache2/
```

- Le fichier **apache2.conf** est le fichier principal de configuration, c'est à partir de celui-ci que tous les autres fichiers sont chargés.
- Le fichier **sites-available** contient la liste des hôtes virtuels installés et le fichier **sites-enabled** celle des hôtes virtuels activés.

Pour plus de détails sur les autres fichiers de configuration du serveur web apache, [cliquez ici](#).

## 2 Serveur HTTP dynamique avec express.js

### [Repository GitHub Task 2](#)

La première étape consistait à créer une image Docker. Pour cela, nous avons utilisé l'image officielle **Node.js**. Nous avons spécifié un programme à démarrer lors de l'exécution d'un conteneur possédant cette image comme origine.

```
FROM node:4.4

LABEL maintainer Tano Iannetta <tano.iannetta@heig-vd.ch>
LABEL maintainer Loan Lassalle <loan.lassalle@heig-vd.ch>

COPY ./src/ /opt/app/

CMD ["node", "/opt/app/index.js"]
```

Une fois l'image créée, nous avons généré le fichier package.json, permettant d'apporter des informations supplémentaires sur l'image précédente. Nous avons utilisé la commande suivante :

```
npm init
```

Fichier package.json généré par la commande précédente

```
{
  "name": "addresses",
  "version": "0.1.0",
  "description": "Just for a demo",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Loan Lassalle",
  "license": "ISC"
}
```

Ensuite, nous avons implémenté le programme permettant de générer, de manière dynamique et aléatoire, du contenu au format JSON.

Afin de générer des données aléatoires au format JSON, nous avons décidé d'utiliser le module npm chance. Pour cela, nous avons enregistré le module npm chance dans les dépendances du programme.

```
npm install --save chance
```

Pour mettre en place un serveur HTTP dynamique, nous avons utilisé le framework `express.js`. Simple d'utilisation, il nous a permis d'implémenter rapidement un serveur HTTP capable de répondre à des requêtes GET.

```
npm install --save express
```

Ensuite, nous avons implémenté le programme au sein du fichier `index.js`. Nous avons décidé de générer des adresses contenant un nom de rue, un code postal, une ville et un pays.

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

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

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

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

```
});

function generateAddresses() {
    var numberOfAddresses = chance.integer({
        min: 0,
        max: 10
    });
    console.log(numberOfAddresses);

    var addresses = [];
    for (var i = 0; i < numberOfAddresses; ++i) {
        addresses.push({
            street: chance.address(),
            postal: chance.postal(),
            city: chance.city(),
            country: chance.country({
                full: true
```

```

        })
    });
};
console.log(addresses);
return addresses;
}

```

Pour finir, nous avons construit l'image Docker et exécuter le conteneur.

```
docker build -t res/express_addresses ./docker-images/express-image
```

```
docker run -d --name express_dynamic -p 9091:3000 res/express_addresses
```

Afin de vérifier le bon fonctionnement du serveur HTTP dynamique, nous avons implémenter des tests. Ils nous ont permis de vérifier la construction des contenus JSON et de vérifier le StatusCode de la réponse du serveur HTTP dynamique.

```

[
  {
    "street": "827 Vuen Turnpike",
    "postal": "K2P 0D6",
    "city": "Buirin",
    "country": "Cape Verde"
  },
  {
    "street": "469 Armav Terrace",
    "postal": "L5A 7M3",
    "city": "Visemolu",
    "country": "Nigeria"
  },
  {
    "street": "1455 Jedkit Square",
    "postal": "X2N 3Q9",
    "city": "Wewhedef",
    "country": "St. Pierre & Miquelon"
  },
  {
    "street": "70 Mirev Parkway",
    "postal": "X1I 6V7",
    "city": "Hicsejmo",
    "country": "St. Lucia"
  }
]

```

```

[
  {
    "street": "443 Jicij Key",
    "postal": "X8L 5A2",
    "city": "Tolfaskok",
    "country": "Morocco"
  },
  {
    "street": "800 Kelok Street",
    "postal": "J7M 4V2",
    "city": "Lopilrem",
    "country": "Barbados"
  },
  {
    "street": "1950 Ziozi Court",
    "postal": "R7Q 1W2",
    "city": "Kitaba",
    "country": "Vietnam"
  },
  {
    "street": "662 Ulopij Place",
    "postal": "B0C 6T9",
    "city": "Paovvi",
    "country": "Latvia"
  }
]

```

Figure 2 : Réponses du serveur HTTP dynamique au format JSON



```

var schema = {
  "type": "object",
  "properties": {
    "street": {
      "type": "string"
    },
    "postal": {
      "type": "string"
    },
    "city": {
      "type": "string"
    },
    "country": {
      "type": "string"
    }
  },
  "required": ["street", "postal", "city", "country"]
};

var data1 = {
  "street": "125 Ifwa Manor",
  "postal": "T3M 3A7",
  "city": "Ivgivep",
  "country": "Guinea"
};

var data2 = {
  "street": 4477,
  "postal": "H9P 8I8",
  "city": "Vomihiu"
};

tests["Valid Data1"] = tv4.validate(data1, schema);
tests["Valid Data2"] = !tv4.validate(data2, schema);
console.log("Validation failed: ", tv4.error);

tests["Status code is 200"] = responseCode.code === 200;

```

Figure 3 : Tests effectués pour la vérification de la réponse du serveur HTTP dynamique

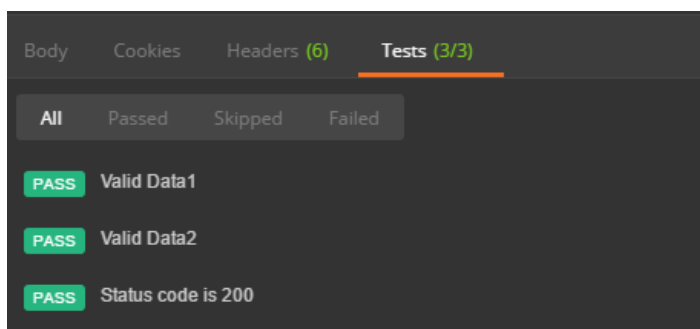


Figure 4 : Validation des tests

A la fin de cette étape, nous avons réussi à mettre en place un serveur dynamique exécutant un script Javascript au sein de notre infrastructure HTTP.

### 3 Proxy inverse avec apache (configuration statique)

#### [Repository GitHub Task 3](#)

La première étape consistait à créer une image Docker. Pour cela, nous avons utilisé l'image officielle PHP. Nous avons spécifié des commandes à exécuter une fois le conteneur démarré.

Le premier module permet d'activer le module **proxy** et **proxy\_http**. Le module **proxy** permet l'implémentation d'un serveur mandataire/passerelle et le module **proxy\_http** apporte le support des requêtes HTTP et HTTPS au module **proxy**.

```
FROM php:7.0-apache
```

```
LABEL maintainer Tano Iannetta <tano.iannetta@heig-vd.ch>
```

```
LABEL maintainer Loan Lassalle <loan.lassalle@heig-vd.ch>
```

```
COPY ./conf/ /etc/apache2/
```

```
RUN a2enmod proxy proxy_http && a2ensite 000-* 001-*
```

Par la suite, nous avons créé deux fichiers de configuration d'hôtes virtuels dans le répertoire **conf/sites-available/**. L'hôte virtuel par défaut du serveur web Apache est défini par le fichier **000-default.conf**. Nous l'avons laissé vide afin de conserver une configuration de base pour le proxy inverse. Le fichier **001-reverse-proxy.conf** définit le proxy inverse. Il permet de rediriger des requêtes effectuées au proxy inverse vers d'autres serveur web, en fonction du nom de domaine et du chemin ciblé.

Grâce à la définition du fichier Dockerfile, le contenu du répertoire **conf** est copié vers le répertoire **/etc/apache2/** de la machine cible afin de configurer les hôtes virtuels du proxy inverse.

Fichier 000-default.conf

```
<VirtualHost *:80>
```

```
</VirtualHost>
```

Le fichier **001-reverse-proxy.conf** contient la définition d'un nom de domaine cible (ServerName), la correspondance entre le chemin cible et l'adresse du serveur destinataire (ProxyPass et ProxyPassReverse)

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

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

    ProxyPass "/api/addresses/" "http://172.17.0.3:3000/"
    ProxyPassReverse "/api/addresses/" http://172.17.0.3:3000/

    ProxyPass "/" "http://172.17.0.2:80/"
    ProxyPassReverse "/" "http://172.17.0.2:80/"
</VirtualHost>
```

Afin de pouvoir contacter le proxy inverse avec le nom de domaine, il a été nécessaire d'ajouter une ligne au sein du fichier **hosts** afin de faire correspondre l'adresse IP **192.168.99.100** de la machine virtuelle Docker et le domaine **demo.res.ch**.

```
# RES laboratoire HTTP Infra
192.168.99.100    demo.res.ch
```

Pour finir, nous avons construit l'image Docker et exécuté le conteneur avec le port mapping.

```
docker build -t res/apache_rp ./docker-images/apache-reverse-proxy
```

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



Figure 5 : Page d'accueil du serveur web Apache à travers le proxy inverse



Figure 6 : Réponse JSON du serveur HTTP dynamique à travers le proxy inverse

A la fin de cette étape, nous avons réussi à mettre en place un proxy inverse statique afin d'obtenir un seul point d'entrée pour notre infrastructure HTTP.

### 3.1 Pourquoi les serveurs ne peuvent pas être atteints directement ?

Les serveurs ne peuvent pas être atteints directement parce que nous n'avons pas défini le mapping des ports de la machine Docker avec ceux des conteneurs, lors de leur exécution. Cependant, si nous avions fait ceci, il aurait été impossible d'exécuter le conteneur du proxy inverse dû à l'utilisation d'un port (port 80) par deux conteneurs.

L'utilisation du proxy inverse permet d'avoir un seul point d'entrée pour d'autres serveurs web. Cela apporte de la simplicité dans la mise en place et dans l'accès aux ressources.

### 3.2 Pourquoi la configuration statique est-elle fragile et doit être améliorée ?

La configuration statique est fragile car elle se base sur les adresses IP des serveurs web saisies en « dur » dans le fichier **001-reverse-proxy.conf**. Si ces adresses IP changent pour une raison ou pour une autre, il sera impossible pour le proxy inverse de contacter les serveurs web cibles.

Cette configuration doit être améliorée afin d'éviter l'action d'un administrateur pour modifier les adresses IP des serveurs web. Cela permettra au proxy inverse d'être flexible et évolutif.

## 4 Requêtes AJAX avec JQuery

### [Repository GitHub Task 4](#)

Dans cette étape, nous avons effectué aucune modification des fichiers Dockerfile.

Dockerfile de l'image Docker res/apache\_php

```
FROM php:7.0-apache

LABEL maintainer Tano Iannetta <tano.iannetta@heig-vd.ch>
LABEL maintainer Loan Lassalle <loan.lassalle@heig-vd.ch>

COPY ./content/ /var/www/html/
```

```
docker build -t res/apache_php ./docker-images/apache-php-image
```

Dockerfile de l'image Docker res/express\_addresses

```
FROM node:4.4

LABEL maintainer Tano Iannetta <tano.iannetta@heig-vd.ch>
LABEL maintainer Loan Lassalle <loan.lassalle@heig-vd.ch>

COPY ./src/ /opt/app/

CMD ["node", "/opt/app/index.js"]
```

```
docker build -t res/express_addresses ./docker-images/express-image
```

Dockerfile de l'image Docker res/apache\_rp

```
FROM php:7.0-apache

LABEL maintainer Tano Iannetta <tano.iannetta@heig-vd.ch>
LABEL maintainer Loan Lassalle <loan.lassalle@heig-vd.ch>

COPY ./conf/ /etc/apache2/

RUN a2enmod proxy proxy_http && a2ensite 000-* 001-*
```

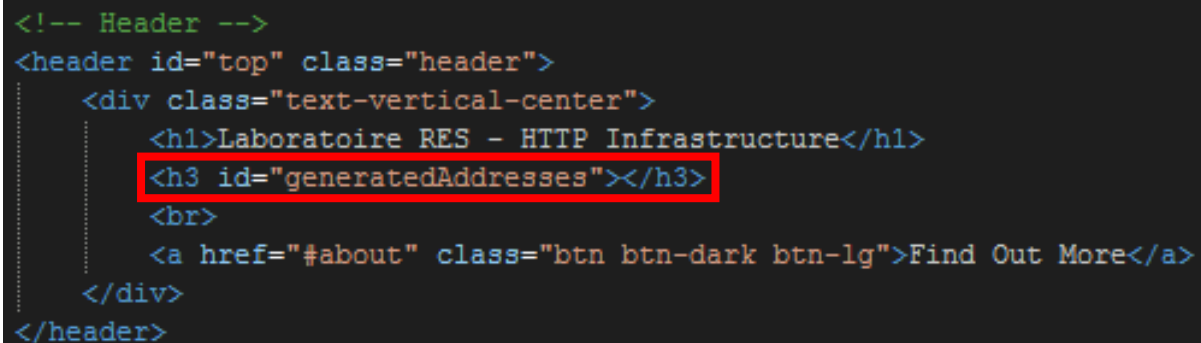
```
docker build -t res/apache_rp ./docker-images/apache-reverse-proxy
```

Par la suite, nous avons exécuté les conteneurs.

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

La deuxième étape consistait à créer un script Javascript permettant d'insérer une adresse dans la page d'accueil du site Internet et d'effectuer un changement dynamique de cette adresse.

Dans un premier temps, nous avons ajouté une balise `<h3>` avec l'id **generatedAddresses** afin de pouvoir spécifier au script l'emplacement de l'insertion de l'adresse générée.



```
<!-- Header -->
<header id="top" class="header">
  <div class="text-vertical-center">
    <h1>Laboratoire RES - HTTP Infrastructure</h1>
    <h3 id="generatedAddresses"></h3>
    <br>
    <a href="#about" class="btn btn-dark btn-lg">Find Out More</a>
  </div>
</header>
```

Figure 7 : Emplacement de l'adresse au sein du fichier index.html

Nous avons aussi ajouté la référence au script Javascript à la fin du fichier index.html.

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

Le script Javascript que nous avons implémenté effectue une requête au proxy inverse vers l'URL **/api/addresses** afin de récupérer les adresses générées au format JSON. Une fois que les caractéristiques de l'adresse cible ont été récupérées, nous affectons les chaînes de caractères à la balise qui possède l'id **generatedAddresses**.

Pour finir, nous paramétrons un intervalle d'exécution afin que l'adresse de la page d'accueil change régulièrement.

```
$(function() {
  console.log("Loading addresses");

  function loadAddresses() {
    $.getJSON("/api/addresses/", function(addresses) {
```

```
console.log(addresses);  
var message = "No address";  
  
if (addresses.length > 0) {  
    message = addresses[0].street + " "  
        + addresses[0].city + " "  
        + addresses[0].country;  
}  
$("#generatedAddresses").text(message);  
});  
}  
  
loadAddresses();  
setInterval(loadAddresses, 1000);  
});
```

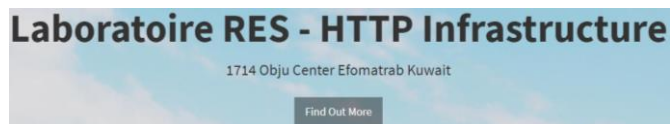
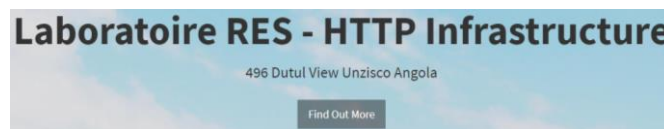
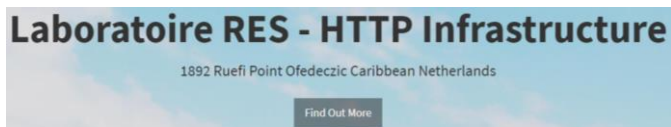


Figure 8 : Page d'accueil avec différentes adresses générées toutes les 2 secondes

A la fin de cette étape, nous avons réussi à mettre en place un site Internet avec du contenu dynamique.

#### 4.1 Est-ce que les demandes AJAX sont envoyées par le navigateur ?

Les demandes AJAX sont envoyées par le navigateur parce que le script Javascript est intégré à la page d'accueil. Lors de l'envoi de la page d'accueil, le client exécute le script une première fois avant de paramétrer son exécution à intervalle régulier (voir ci-dessus). Ce script exécute une requête AJAX toutes les 2 secondes afin de demander une nouvelle adresse aléatoire au serveur.

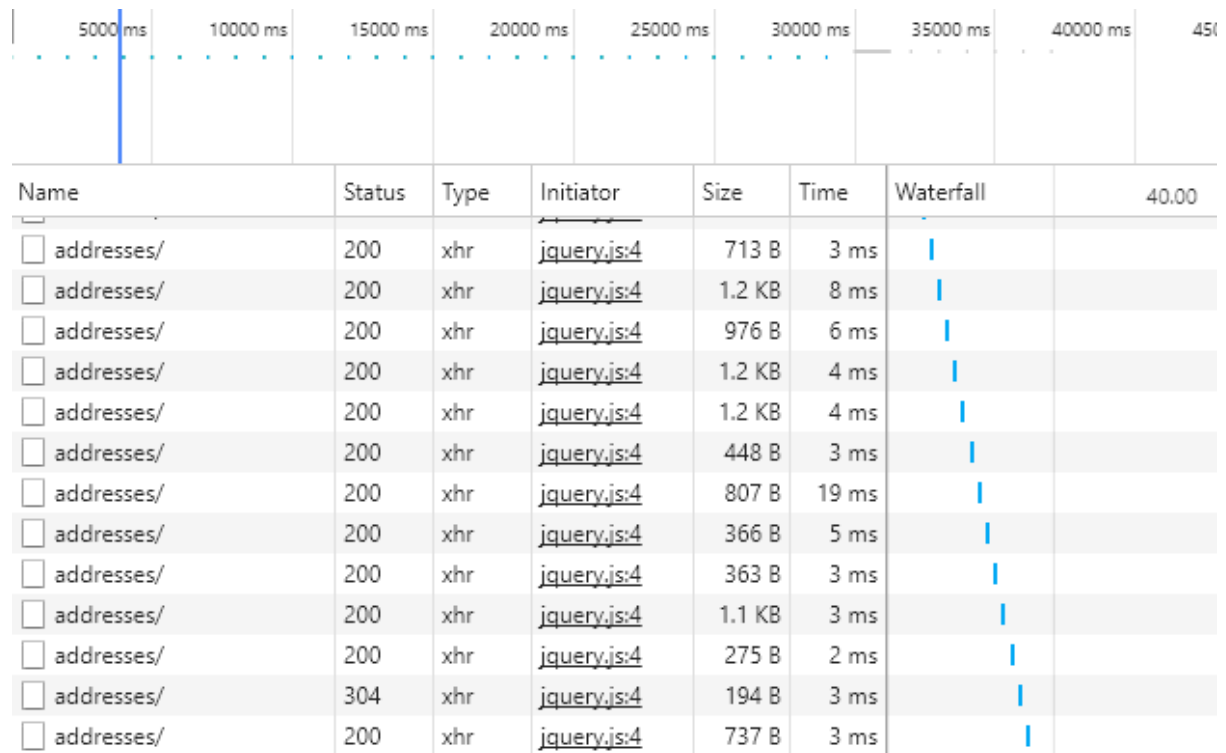


Figure 9 : Requête effectuée par le navigateur toutes les 2 secondes

#### 4.2 Pourquoi la démonstration ne fonctionnerait pas sans un proxy inverse ?

La démonstration ne fonctionnerait pas sans un proxy inverse parce que la same-origin policy empêche l'accès en lecture et en écriture, à des serveurs dont le nom de domaine est différent de celui d'où provient la page en cours.



## 5 Configuration dynamique du proxy inversé

### [Repository GitHub Task 5](#)

La première étape consistait à d'obtenir une configuration beaucoup plus flexible et évolutive par rapport aux adresses IP affectées au serveur Internet.

Pour cela nous avons créé un script bash qui remplace les chaînes de caractères `{{ip_apache_static}}` et `{{ip_express_dynamic}}` par les valeurs des variables fournies lors de l'exécution du conteneur. Lors de l'exécution du conteneur, les variables `IP_APACHE_STATIC` et `IP_EXPRESS_DYNAMIC` seront remplacées par leur valeur.

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

# Set ip addresses
CONF_FILE="/etc/apache2/sites-available/001-reverse-proxy.conf"

sed -i "s/{{ip_express_dynamic}}/$IP_EXPRESS_DYNAMIC/g" $CONF_FILE
sed -i "s/{{ip_apache_static}}/$IP_APACHE_STATIC/g" $CONF_FILE

# Start apache2
apache2 -DFOREGROUND
```

Ces chaînes de caractères spécifiques ont été insérées dans le fichier de configuration du proxy inverse **001-reverse-proxy.conf** afin de pouvoir modifier les adresses IP des serveurs.

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

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

    ProxyPass '/api/addresses/' 'http://{{ip_express_dynamic}}/'
    ProxyPassReverse '/api/addresses/' 'http://{{ip_express_dynamic}}/'

    ProxyPass '/' 'http://{{ip_apache_static}}/'
    ProxyPassReverse '/' 'http://{{ip_apache_static}}/'
</VirtualHost>
```

Ensuite, il suffit d'indiquer à l'image Docker **res/apache\_rp** de copier le script et de l'exécuter.

```
FROM php:7.0-apache

LABEL maintainer Tano Iannetta <tano.iannetta@heig-vd.ch>
LABEL maintainer Loan Lassalle <loan.lassalle@heig-vd.ch>

COPY ./conf/ /etc/apache2/
COPY ./apache2-foreground.sh /opt/app/apache2-foreground.sh

ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data
ENV APACHE_LOG_DIR /var/log/apache2
ENV APACHE_LOCK_DIR /var/lock/apache2
ENV APACHE_PID_FILE /var/run/apache2.pid

RUN a2enmod proxy proxy_http && a2ensite 000-* 001-*

CMD ["sh", "/opt/app/apache2-foreground.sh"]
```

Puis construire à nouveau l'image Docker du proxy inverse.

```
docker build -t res/apache_rp ./docker-images/apache-reverse-proxy
```

Pour terminer, afin de prouver que l'insertion des adresses IP est fonctionnelle, nous avons exécuté plusieurs conteneurs de chaque serveur Internet afin d'affecter des adresses IP différentes aux précédentes étape du laboratoire. Nous avons récupéré l'adresse IP des serveurs non factices afin d'indiquer au proxy inverse les serveurs cibles pour le contenu statique et dynamique.

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

```
docker run -d --name apache_static res/apache_php
docker inspect $1 | grep "\"IPAddress\"" -m 1 | grep -o "[0-9.]\+"
```

```
docker run -d --name express_dynamic res/express_addresses
docker inspect $1 | grep "\"IPAddress\"" -m 1 | grep -o "[0-9.]\+"
```

```
docker run -d --name apache_rp -p 8080:80 -e  
IP_APACHE_STATIC=<ip_address:port> -e  
IP_EXPRESS_DYNAMIC=<ip_address:port> res/apache_rp
```

Lors de l'exécution, il est possible d'obtenir l'erreur suivante : **/opt/app/apache2-foreground.sh: 2: set: Illegal option -**.

Cette erreur est due au style des sauts de ligne (CRLF). Pour corriger ce problème, il est nécessaire de convertir les sauts de ligne CRLF en LF.

```
sed -i "s/\r$//" ./docker-images/apache-reverse-proxy/apache2-foreground.sh
```

A la fin de cette étape, nous avons réussi à mettre en place un proxy inverse dynamique au sein de notre infrastructure HTTP.

## 6 Etapes additionnelles

La plupart des étapes supplémentaires ont été résolues par l'utilisation de Traefik.

Traefik est un proxy HTTP inverse et un équilibreur de charge conçu pour déployer des microservices avec facilité. Il supporte plusieurs backends, dont Docker, pour gérer sa configuration automatiquement et dynamiquement. Il peut écouter le socket docker pour détecter la présence, le démarrage ou l'arrêt des conteneurs et se configurer en conséquence.

Traefik prend en charge l'utilisation de l'étiquette du conteneur comme directive de configuration. Ces étiquettes sont définies dans les fichiers Dockerfiles.

La configuration de Traefik s'effectue par l'utilisation du fichier `/etc/traefik/traefik.toml`.

A la fin de cette étape, nous avons réussi à mettre en place un proxy inverse dynamique capable d'effectuer de la répartition de charge au sein de notre infrastructure HTTP.

### 6.1 Répartition de charge

[Repository GitHub Task 6.1](#)

Pour qu'un conteneur s'annonce comme un service particulier, nous avons défini une étiquette **traefik.backend**. Si le port de ce service est différent de 80, il est nécessaire de le configurer avec l'étiquette **traefik.port**. Pour terminer, l'étiquette **traefik.frontend.rule**, indique la requête HTTP qui doit être transmise au service.

Au sein de notre configuration, nous avons modifié des fichiers Dockerfile afin de définir les services de notre infrastructure.

Pour le service fournissant le site Internet statique, nous avons défini qu'il devait répondre aux requêtes possédant le nom d'hôte **demo.res.ch** et le chemin **/**. Ces étiquettes indiquent à Traefik que la requête est adressée à un conteneur exécutant une instance du service **apache\_static**.

Dockerfile de l'image Docker res/apache\_php

```
FROM php:7.0-apache

LABEL maintainer Tano Iannetta <tano.iannetta@heig-vd.ch>
LABEL maintainer Loan Lassalle <loan.lassalle@heig-vd.ch>

LABEL "traefik.backend"="apache_static"
LABEL "traefik.frontend.rule"="Host: demo.res.ch;PathPrefix: /"

COPY ./content/ /var/www/html/
```

Pour le service fournissant le site les adresses aléatoires dynamiques, nous avons défini qu'il devait répondre aux requêtes possédant le nom d'hôte **demo.res.ch** et le chemin **/api/addresses/**. Ces étiquettes indiquent à Traefik que la requête est adressée à un conteneur exécutant une instance du service **express\_dynamic**.

Dockerfile de l'image Docker res/express\_addresses

```
FROM node:4.4

LABEL maintainer Tano Iannetta <tano.iannetta@heig-vd.ch>
LABEL maintainer Loan Lassalle <loan.lassalle@heig-vd.ch>

LABEL "traefik.backend"="express_dynamic"
LABEL "traefik.port"="3000"
LABEL "traefik.frontend.rule"="Host: demo.res.ch;PathStrip: /api/addresses/"

COPY ./src/ /opt/app/

CMD ["node", "/opt/app/index.js"]
```

Ensuite, nous avons créé une image Docker de Traefik. Pour cela, nous avons utilisé l'image officielle. Nous avons indiqué à l'image Docker **res/traefik** de copier le fichier de configuration **traefik.toml** dans le répertoire **/etc/traefik/** du conteneur.

```
FROM traefik:latest

LABEL maintainer Tano Iannetta <tano.iannetta@heig-vd.ch>
LABEL maintainer Loan Lassalle <loan.lassalle@heig-vd.ch>

COPY ./src/traefik.toml /etc/traefik/traefik.toml
```

Nous avons modifié le fichier de configuration traefik.toml afin que Traefik nous affiche les différents conteneurs contactés et active aussi l'interface d'administration des conteneurs en cours d'exécution.

```
#####
# Global configuration
#####

debug = true

#####

# Web configuration backend
#####

[web]
address = ":8080"

#####

# Docker configuration backend
#####

[docker]
endpoint = "unix:///var/run/docker.sock"
domain = "demo.res.ch"
watch = true
```

Pour terminer, afin de vérifier le bon fonctionnement de l'infrastructure avec Traefik, nous avons contruit à nouveau les images Docker utilisées et exécutées un conteneur de chacune de ces images. Puis, pour prouver la répartition des charges, nous avons décidé d'exécuter deux autres conteneurs correspondant aux images Docker `res/apache_php` et `res/express_addresses`.

```
docker build -t res/apache_php ./docker-images/apache-php-image
docker build -t res/express_addresses ./docker-images/express-image
docker build -t res/traefik ./docker-images/traefik
```

```
docker run -d --name apache_static res/apache_php
docker run -d --name express_dynamic res/express_addresses
docker run -d --name traefik -p 9090:8080 -p 8080:80 -v
/var/run/docker.sock:/var/run/docker.sock res/traefik
```

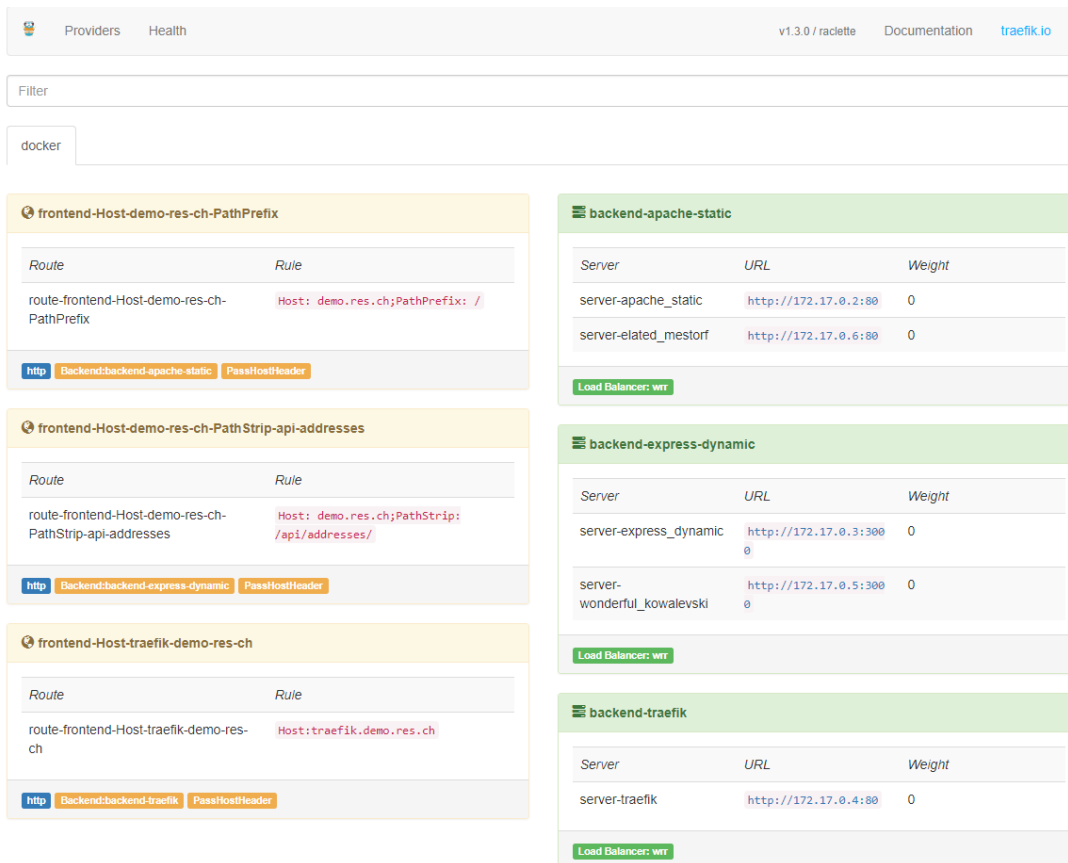
```
docker run -d res/apache_php
docker run -d res/express_addresses
```

Afin de voir quel conteneur répond à une requête, nous avons utilisé les informations de débogage de Traefik.

```
docker attach traefik
```

```
time="2017-06-11T12:25:03Z" level=debug msg="Round trip: http://172.17.0.3:3000, code: 200, duration: 1.540983ms"
time="2017-06-11T12:25:05Z" level=debug msg="Round trip: http://172.17.0.5:3000, code: 200, duration: 9.784323ms"
time="2017-06-11T12:25:06Z" level=debug msg="Round trip: http://172.17.0.2:80, code: 200, duration: 1.342632ms"
time="2017-06-11T12:25:06Z" level=debug msg="Round trip: http://172.17.0.2:80, code: 200, duration: 47.048305ms"
time="2017-06-11T12:25:06Z" level=debug msg="Round trip: http://172.17.0.6:80, code: 200, duration: 78.600101ms"
time="2017-06-11T12:25:06Z" level=debug msg="Round trip: http://172.17.0.2:80, code: 200, duration: 73.815937ms"
time="2017-06-11T12:25:06Z" level=debug msg="Round trip: http://172.17.0.6:80, code: 200, duration: 73.915791ms"
time="2017-06-11T12:25:06Z" level=debug msg="Round trip: http://172.17.0.2:80, code: 200, duration: 1.746268ms"
time="2017-06-11T12:25:06Z" level=debug msg="Round trip: http://172.17.0.6:80, code: 200, duration: 2.282307ms"
time="2017-06-11T12:25:06Z" level=debug msg="Round trip: http://172.17.0.6:80, code: 200, duration: 3.87137ms"
time="2017-06-11T12:25:06Z" level=debug msg="Round trip: http://172.17.0.2:80, code: 200, duration: 1.675454ms"
time="2017-06-11T12:25:06Z" level=debug msg="Round trip: http://172.17.0.6:80, code: 200, duration: 4.499152ms"
time="2017-06-11T12:25:06Z" level=debug msg="Round trip: http://172.17.0.6:80, code: 200, duration: 3.904495ms"
time="2017-06-11T12:25:06Z" level=debug msg="Round trip: http://172.17.0.2:80, code: 200, duration: 16.227642ms"
time="2017-06-11T12:25:06Z" level=debug msg="Round trip: http://172.17.0.2:80, code: 200, duration: 8.020803ms"
time="2017-06-11T12:25:06Z" level=debug msg="Round trip: http://172.17.0.6:80, code: 200, duration: 2.954826ms"
time="2017-06-11T12:25:06Z" level=debug msg="Round trip: http://172.17.0.3:3000, code: 200, duration: 2.607618ms"
time="2017-06-11T12:25:08Z" level=debug msg="Round trip: http://172.17.0.5:3000, code: 200, duration: 3.951986ms"
time="2017-06-11T12:25:10Z" level=debug msg="Round trip: http://172.17.0.3:3000, code: 200, duration: 2.388872ms"
time="2017-06-11T12:25:12Z" level=debug msg="Round trip: http://172.17.0.5:3000, code: 200, duration: 1.101999ms"
time="2017-06-11T12:25:14Z" level=debug msg="Round trip: http://172.17.0.3:3000, code: 200, duration: 5.116865ms"
```

Figure 10 : Informations de débogage de Trafik



The screenshot shows the Traefik dashboard with the following configuration:

- Frontend: frontend-Host-demo-res-ch-PathPrefix**
  - Route: route-frontend-Host-demo-res-ch-PathPrefix
  - Rule: Host: demo.res.ch;PathPrefix: /
  - Backend: backend-apache-static
  - PassHostHeader: true
- Frontend: frontend-Host-demo-res-ch-PathStrip-api-addresses**
  - Route: route-frontend-Host-demo-res-ch-PathStrip-api-addresses
  - Rule: Host: demo.res.ch;PathStrip: /api/addresses/
  - Backend: backend-express-dynamic
  - PassHostHeader: true
- Frontend: frontend-Host-traefik-demo-res-ch**
  - Route: route-frontend-Host-traefik-demo-res-ch
  - Rule: Host:traefik.demo.res.ch
  - Backend: backend-traefik
  - PassHostHeader: true
- Backend: backend-apache-static**
  - Server: server-apache\_static (URL: http://172.17.0.2:80, Weight: 0)
  - Server: server-elated\_mestorf (URL: http://172.17.0.6:80, Weight: 0)
  - Load Balancer: wrr
- Backend: backend-express-dynamic**
  - Server: server-express\_dynamic (URL: http://172.17.0.3:3000, Weight: 0)
  - Server: server-wonderful\_kowalevski (URL: http://172.17.0.5:3000, Weight: 0)
  - Load Balancer: wrr
- Backend: backend-traefik**
  - Server: server-traefik (URL: http://172.17.0.4:80, Weight: 0)
  - Load Balancer: wrr

Figure 11 : Accès à la page de contrôle de Traefik avec l'url <http://demo.res.ch:9090/>

## 6.2 Répartition de charge avec affinité de serveur

L'étiquette **traefik.backend.loadbalancer.sticky**, fournie par Traefik, nous donne la possibilité d'activer ou désactiver la répartition de charge avec affinité de serveur « **sticky session** ». Lorsque la valeur de l'étiquette est à true, les « sessions collantes » sont activées. Sinon la répartition de charge s'effectue avec l'algorithme **round-robin**.

Dockerfile de l'image Docker res/apache\_php

```
FROM php:7.0-apache

LABEL maintainer Tano Iannetta <tano.iannetta@heig-vd.ch>
LABEL maintainer Loan Lassalle <loan.lassalle@heig-vd.ch>

LABEL "traefik.backend"="apache_static"
LABEL "traefik.frontend.rule"="Host: demo.res.ch;PathPrefix: /"
LABEL "traefik.backend.loadbalancer.sticky"="true"

COPY ./content/ /var/www/html/
```

Dockerfile de l'image Docker res/express\_addresses

```
FROM node:4.4

LABEL maintainer Tano Iannetta <tano.iannetta@heig-vd.ch>
LABEL maintainer Loan Lassalle <loan.lassalle@heig-vd.ch>

LABEL "traefik.backend"="express_dynamic"
LABEL "traefik.port"="3000"
LABEL "traefik.frontend.rule"="Host: demo.res.ch;PathStrip:
/api/addresses/"
LABEL "traefik.backend.loadbalancer.sticky"="true"

COPY ./src/ /opt/app/

CMD ["node", "/opt/app/index.js"]
```

### 6.3 Gestion dynamique des agrégats de conteneurs

Traefik utilise les événements déclenchés par le socket docker pour se mettre à jour à chaque fois qu'un conteneur a été démarré ou arrêté.

### 6.4 Interface d'administration

Nous avons choisi de ne pas implémenter une interface d'administration permettant d'afficher et de mettre jour notre infrastructure. Toutefois, à travers l'utilisation de Traefik, il est possible d'afficher les différents serveurs Internet en backend en cours d'exécution.

## 7 Procédure de validation

La procédure peut être effectuée à l'aide d'un script créé par nos soins. Il s'occupe de construire les images Docker dont nous avons besoin, d'exécuter les conteneurs et d'afficher les pages Internet permettant de vérifier le bon fonctionnement de l'infrastructure.

Lors de l'exécution du script validate.sh, il est nécessaire de donner un paramètre avec comme valeur « step1 », « step2 », « step3 », « step4 », « step5 », « step61 » ou « step62 »



Voici le script bash utilisé pour valider chaque étape de la mise en place de l'infrastructure HTTP.

```
#!/bin/bash

HOST="demo.res.ch"

function show_help
{
    echo "$1 is not a valid argument"
    echo "The parameter can be
\"step1\", \"step2\", \"step3\", \"step4\", \"step5\", \"step61\" or \"step62\""
}

function stop_all {
    echo "Stopping all launched container"
    docker kill $(docker ps -aq)
    docker rm $(docker ps -aq)
}

function start_step1 {
    local port=9090

    echo "Starting step 1"
    docker build -t res/apache_php ./docker-images/apache-php-image
    docker run -d --name apache_static -p $port:80 res/apache_php
    start "http://$HOST:$port"
}

function start_step2
{
    local port=3000

    echo "Starting step 2"
    docker build -t res/express_addresses ./docker-images/express-image
    docker run -d --name express_dynamic -p $port:3000
res/express_addresses
    start "http://$HOST:$port/"
}
```

```

}

function start_step3 {
    local port=8080

    echo "Starting step 3"
    echo "Do not forget to add $HOST to your hosts file for correct DNS
    resolving."

    docker build -t res/apache_php ./docker-images/apache-php-image
    docker run -d --name apache_static res/apache_php

    docker build -t res/express_addresses ./docker-images/express-image
    docker run -d --name express_dynamic res/express_addresses

    docker build -t res/apache_rp ./docker-images/apache-reverse-proxy
    docker run -d --name apache_rp -p $port:80 res/apache_rp
    start "http://$HOST:$port/"
    start "http://$HOST:$port/api/addresses/"
}

function start_step4 {
    local port=8080

    echo "Starting step 4"
    echo "Do not forget to add $HOST to your hosts file for correct DNS
    resolving."

    docker build -t res/apache_php ./docker-images/apache-php-image
    docker run -d --name apache_static res/apache_php

    docker build -t res/express_addresses ./docker-images/express-image
    docker run -d --name express_dynamic res/express_addresses

    docker build -t res/apache_rp ./docker-images/apache-reverse-proxy
    docker run -d --name apache_rp -p $port:80 res/apache_rp
    start "http://$HOST:$port/"
}

```

```
}

function get_last_container_ip {
    docker inspect $1 | grep "\"IPAddress\"" -m 1 | grep -o "[0-9.]\+"
}

function start_step5 {
    local port=8080

    echo "Starting step 5"
    echo "Do not forget to add $HOST to your hosts file for correct DNS
    resolving."

    docker build -t res/apache_php ./docker-images/apache-php-image
    docker run -d --name apache_static res/apache_php
    local ip_apache_static=$(get_last_container_ip apache_static):80

    docker build -t res/express_addresses ./docker-images/express-image
    docker run -d --name express_dynamic res/express_addresses
    local ip_express_dynamic=$(get_last_container_ip express_dynamic):3000

    sed -i "s/\r$//" ./docker-images/apache-reverse-proxy/apache2-
    foreground.sh

    docker build -t res/apache_rp ./docker-images/apache-reverse-proxy
    docker run -d --name apache_rp -p $port:80 -e
    IP_APACHE_STATIC=$ip_apache_static \
        -e IP_EXPRESS_DYNAMIC=$ip_express_dynamic res/apache_rp
    start "http://$HOST:$port/"

    echo $ip_apache_static $ip_express_dynamic
}

function start_step61
{
    local site_port=8080
    local monitor_port=9090
```

```

echo "Starting supplementary steps"

docker build -t res/apache_php ./docker-images/apache-php-image
docker run -d --name apache_static res/apache_php
docker run -d res/apache_php

docker build -t res/express_addresses ./docker-images/express-image
docker run -d --name express_dynamic res/express_addresses
docker run -d res/express_addresses

docker build -t res/traefik ./docker-images/traefik
docker run -d --name traefik -p $monitor_port:8080 -p $site_port:80 -v
/var/run/docker.sock:/var/run/docker.sock res/traefik

start "http://$HOST:$site_port/"
start "http://$HOST:$monitor_port/"
docker attach traefik
}

# Main
error=false

case $1 in
    "step1" ) start_step1
                ;;
    "step2" ) start_step2
                ;;
    "step3" ) start_step3
                ;;
    "step4" ) start_step4
                ;;
    "step5" ) start_step5
                ;;
    "step61"|"step62" ) start_step61
                ;;
    * ) error=true; show_help

```

```
esac

read -p "Press any key to close all docker container ..."

if ! $error
then
    stop_all
fi
```

## 8 Commandes diverses

### 8.1 Supprime tous les conteneurs

```
docker kill $(docker ps -aq)
docker rm $(docker ps -aq)
```

### 8.2 Supprime toutes les images

```
docker rmi $(docker images -aq)
```

### 8.3 Accède au terminal de la machine virtuel Docker

```
docker-machine ssh
```