11/06/2017

Rapport RES

Infrastructure HTTP

Xavier Vaz Afonso & Adrian Di Pietro

Table des matières

Introduction

Ce laboratoire consiste à mettre en place une infrastructure web. Nous avons utilisé docker pour la création des containers qui contiennent les différents serveurs qui auront chacun un rôle spécifique.

Voici le lien de notre repo sur github : https://github.com/xav21/Teaching-HEIGVD-RES-2017-Labo-HTTPInfra

Étape 1

Pour cette étape nous avons créé une nouvelle branche qui s'appelle « fb-apache-static ».

L'objectif de cette étape était de mettre en place une infrastructure pour un serveur HTTP. Pour cela nous avons utilisé un Dockerfile. Il contient toutes les instructions pour la création d'une image. Dans notre cas on se base sur une image déjà existante.

Pour finir nous avons ajouté un template de site web déjà créé (basé sur bootstrap) pour vérifier que notre solution fonctionne.

Notre Dockerfile se base sur l'image officielle de PHP. En effet cette image contient un serveur apache déjà configuré ce qui est très pratique pour ce projet. Nous avons utilisé la version 7.0 d'apache qui est la dernière version en ligne.

Nous avons ensuite téléchargé un template bootstrap existant et nous l'avons placé dans le répertoire qui s'appelle « content ». Le fichier index.html permet d'accéder à la page d'accueil du site, c'est d'ailleurs ce fichier qui est choisi par défaut comme point d'entrée.

Voici le contenu de note fichier DockerFile :

```
FROM php: 7.0-apache

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

Au final tout ce que fait notre Dockerfile est de se baser sur l'image PHP et de copier le contenu du répertoire « content » dans le répertoire de « /var/www/HTML » du container.

Les fichiers configurations d'apache sont situés dans le répertoire suivant : « /etc/apache2/ »

Ce répertoire contient un dossier s'appelant « sites-available ». C'est à cet endroit qu'on trouve les spéciations des sous-configurations. En effet un serveur apache peut héberger plusieurs sites web.

Pour construire notre image : docker build -t res/apache_php .

Pour créer un container basé sur notre image : docker run -d -p 9090:80 res/apache_php

On a également mappé le port 80 du container sur notre port 9090 (paramètre –p 9090 :80).

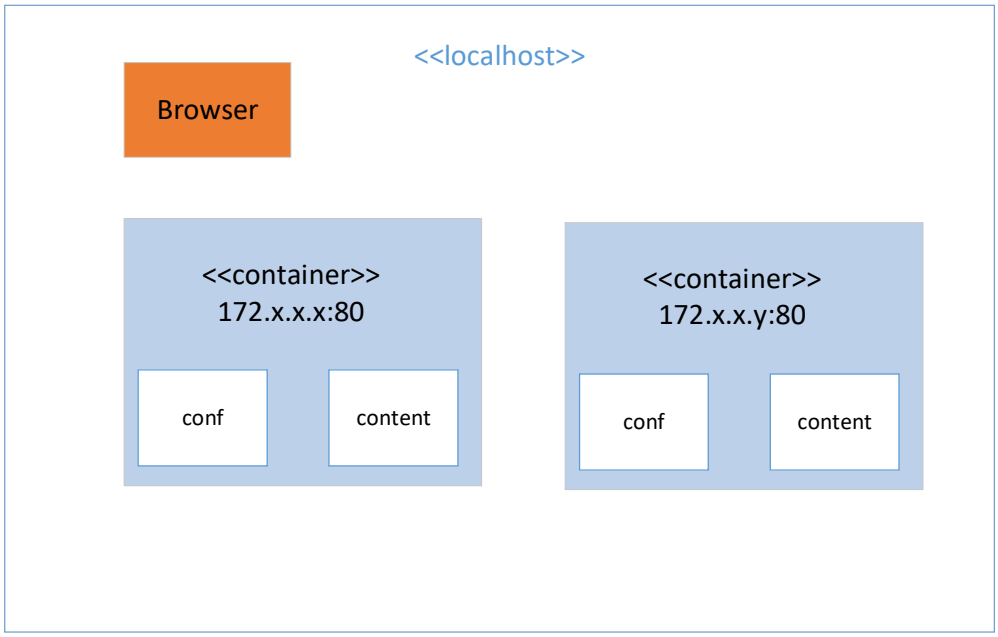
Pour tuer un container : docker kill « nom du container »

Pour pouvoir accéder à cet emplacement, il faut utiliser la commande suivante.

Docker exec –it « nom du container » /bin/bash

Cette commande permet de se connecter sur un container en exécution et ouvrir un terminal pour explorer le contenu.

Notre configuration



Étape 2

Pour cette étape nous avons créé une nouvelle branche qui s'appelle « fb-express-dynamic » et qui se base directement sur la branche précédente « fb-apache-static ».

L'objectif de cette étape était de créer une application Node.js qui retourne un tableau JSON généré aléatoirement à chaque requête de type « GET » (contenu dynamique). Pour ce faire, nous avons créé une nouvelle image pour réaliser cette étape.

Voici le contenu de note fichier DockerFile :

```
FROM node:6.10

COPY src /opt/app

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

On se base sur l'image contenant la version 6.10 de node.js. C'est la version la plus stable actuellement.

On copie tous le contenu du fichier « src » dans le répertoire « /opt/app » du container.

On exécute ensuite le script « index.js.»

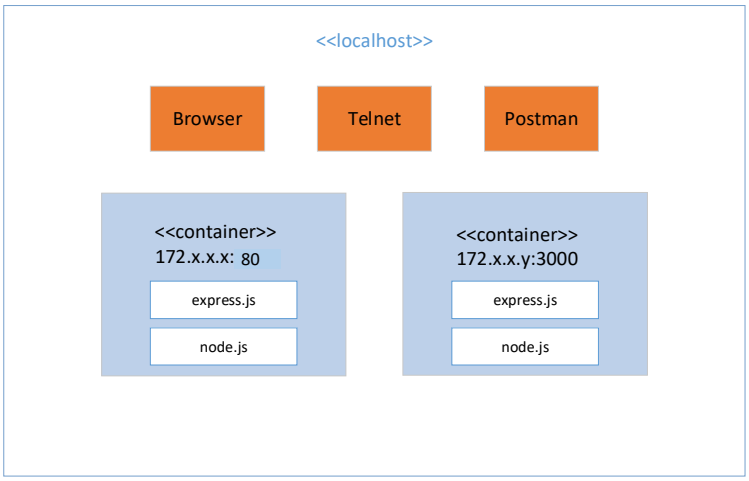
Ce script va générer une liste d'emplacement allant de 0 à 10. Nous avons utilisé la librairie « Chance.js » pour générer aléatoirement ces emplacements et la librairie « express.js » pour la création très rapide d'un serveur HTTP avec node.js.

Durant cette étape nous avons également dû créer un fichier « package.json » pour gérer les dépendances avec les librairies utilisées.

Pour créer ce document, il suffit d'utiliser ces commandes :

npm init (répondre aux questions de base)
npm install –save chance
npm install –save express

Notre configuration :



Étape 3

Pour cette étape nous avons créé une nouvelle branche qui s'appelle « fb-apache-reverse-proxy» et qui se base directement sur la branche précédente «fb-express-dynamic ».

On ne peut pas atteindre les serveurs « static » et « dynamic » car on ne mappe pas leur numéro de port. Ils sont donc inaccessibles depuis l'extérieur. C'est très bien pour la sécurité et ça va être pratique pour la répartition de tâche par la suite.

Cette configuration est fragile car lors de la création d'un container, Docker attribue une nouvelle adresse IP disponible. Avec notre solution actuelle, on a hardcodé des adresses IP, mais ça ne sera pas forcément toujours les mêmes. Il faut donc trouver manière plus dynamique comme solution.

Voici le contenu de note fichier DockerFile :

```
FROM php:7.0-apache

COPY conf/ /etc/apache2/sites-available

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

Ce DockerFile se base sur l'image officielle de PHP.

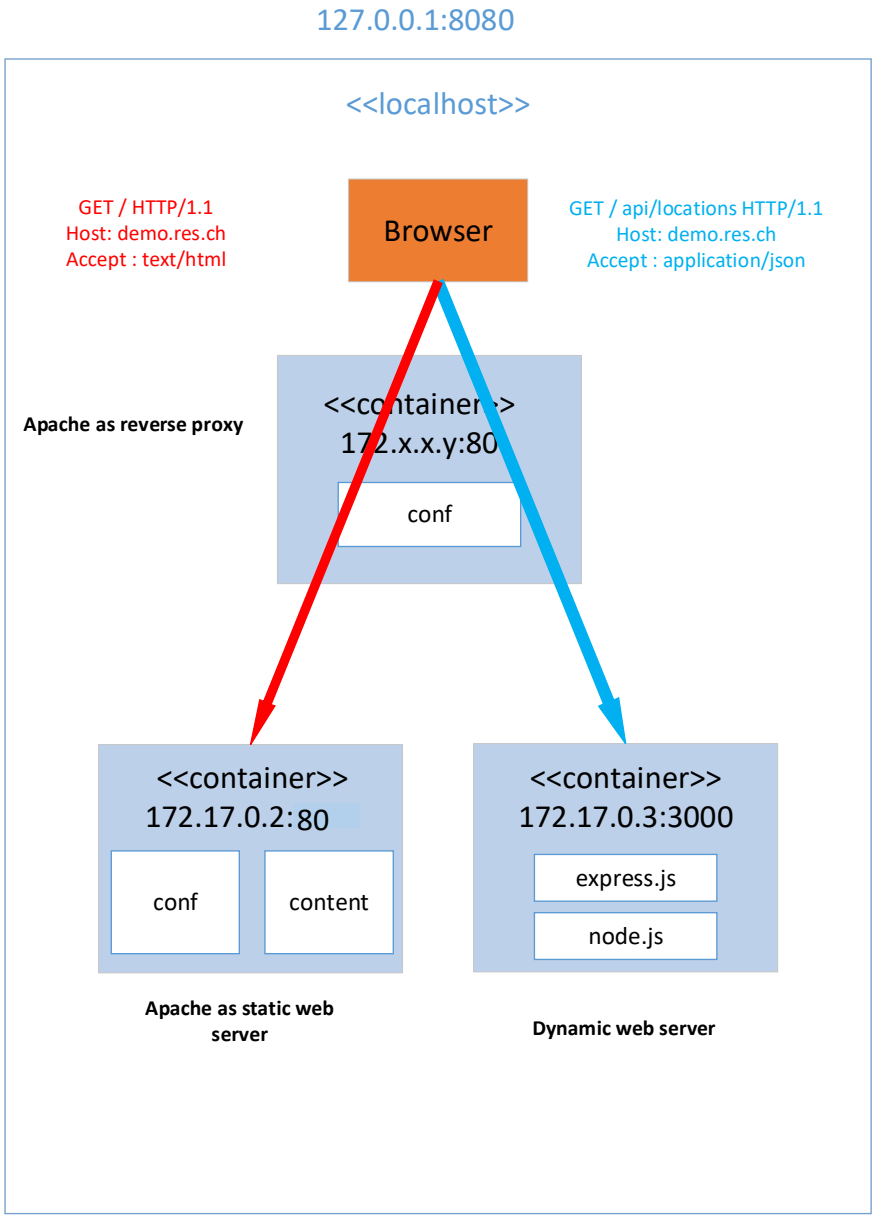
On va copier les configurations se trouvant dans le dossier « conf » qu'on va copier dans le dossier « conf/ /etc/apache2/sites-available ».

C'est le fichier de configuration s'appelant « 001-reverse-proxy.conf ». Son rôle est de savoir où renvoyer les requêtes.

On va activer les modules proxy et proxy_http qui sont obligatoires lorsqu'on décide de créer un serveur de ce type.

On active la configuration par défaut et celui de notre site web.

Notre configuration pour les étapes 3 et 4 :



Étape 4

Pour cette étape nous avons créé une nouvelle branche qui s'appelle « fb-ajax» et qui se base directement sur la branche précédente «fb-apache-reverse-proxy ».

Dans cette étape, nous avons implémenté des requêtes AJAX. Pour cela nous avons utilisé la librairie JavaScript qui s'appelle JQuery. Ceci nous permet de réactualiser une partie du DOM.

Le serveur apache_php :

```
FROM php:7.0-apache

RUN apt-get update && \
 apt-get install -y vim

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

On met à jour la liste des paquetages pour pouvoir installer le logiciel vim.

Même chose pour le serveur express_dynamic et apache_rp

Pour ajouter notre document JavaScript, il suffit d'ajouter le lien dans le code source de la page HTML.

```
<script src="js/locations.js"></script>
```

```
$(function(){
	function loadLocations(){
		$.getJSON( "/api/locations/", function(locations) {
			console.log(locations);
			var message = "no result";
			if(locations.length>0){
				message = locations[0].country + " " + locations[0].address;
			}
			$(".test").text(message);
		});
	};
	loadLocations();
	setInterval(loadLocations,2000);
})
```

Ce script permet de faire des requêtes HTTP périodiquement pour récupérer une liste d'emplacements. On choisit le premier élément de cette liste qu'on va insérer dans une balise que nous avons appelée « test ».Ceci permet comme expliqué précédemment de réactualiser une partie du DOM, dans notre cas ça va être le contenu des balises ayant un classe avec le nom « test ».

Étape 5

Pour cette étape nous avons créé une nouvelle branche qui s'appelle « fb-dynamic-configuration» et qui se base directement sur la branche précédente «fb-ajax ».

Toutes les étapes précédentes nous ont permis de mettre en place notre infrastructure. Cependant jusqu'à présent nous étions dépendants de certaines adresses IP hardcodé. Ce qui fait qu'on devait s'assurer que lors de la création des containers, ils aient les bonnes adresses IP.

Dans cette étape, nous avons donc remplacé cette configuration pour rendre cette solution dynamique

La solution est de passé par les variables d'environnements grâce à la balise –e lors de la création de containers.

Le fichier apache2-foreground est un document qui s'exécute avant le lancement du serveur apache. Il nous permet de récupérer les informations passées en paramètres et d'appeler un script PHP qui va générer le nouveau fichier de configuration qu'on va ensuite placer dans le dossier de configuration apache.

```
#!/bin/bash
set –e

#Add setup for RES lab
echo "Setup for the RES lab..."
echo "Static app URL: $STATIC_APP"
echo "Dynamic app URL: $DYNAMIC_APP"
php /var/apache2/templates/config-template.php > /etc/apache2/sites-available/001-
reverse-proxy.conf
```

Voici le script PHP qui permet dynamiquement de changer les adresses IP grâce aux variables d'environnement.

```
<?php
        $dynamic_app = getenv('DYNAMIC_APP');
        $static_app = getenv('STATIC_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 "$static_app"?>/'
        ProxyPassReverse '/' '<?php print "$static_app"?>'
</VirtualHost>
```

Pour s'assurer que notre implémentation fonctionne, il faut démarrer plusieurs containers ne servant à « rien ». Ces containers permettent de s'assurer que lors de la création de nos containers basés sur l'image d'apache_php et express_locations d'avoir une autre adresse IP que ceux par défaut.

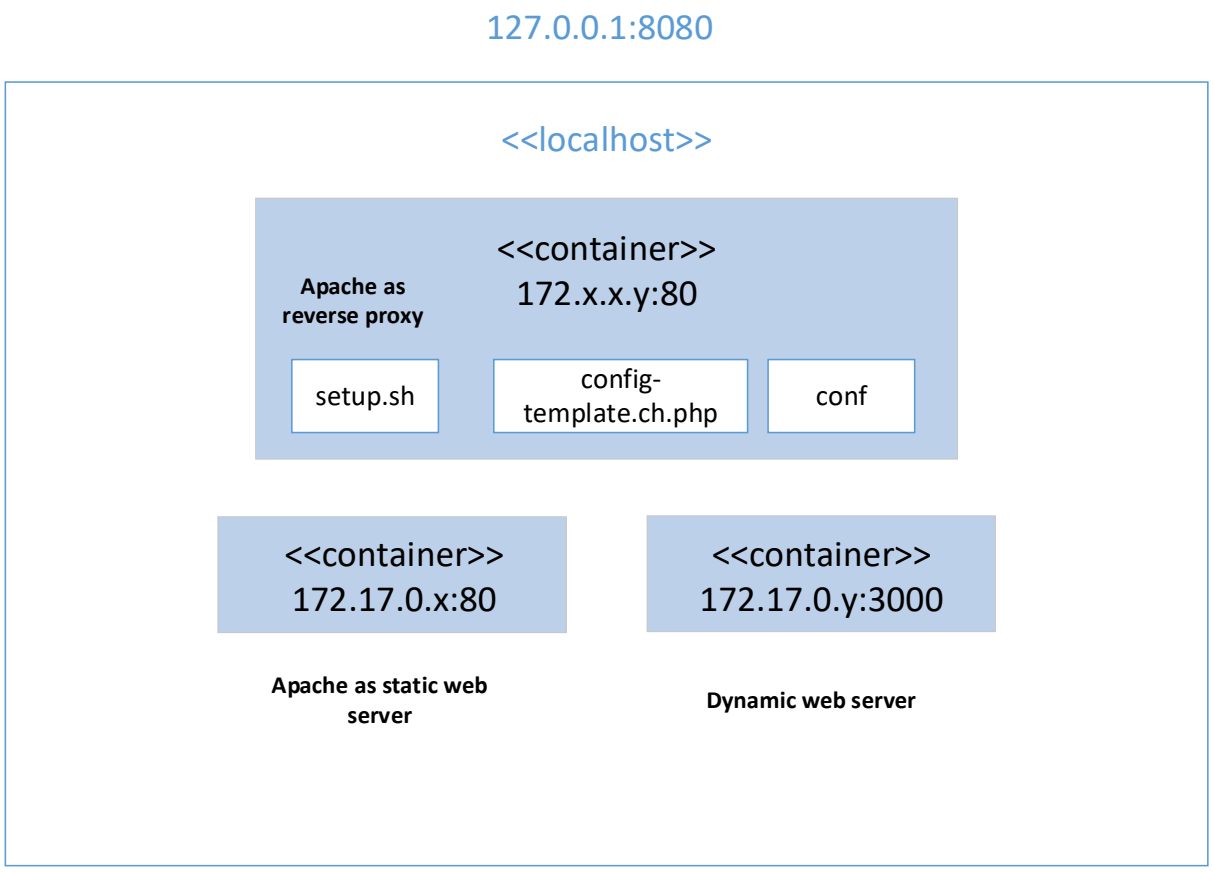
Il suffit ensuite de regarder leur adresse IP avec la commande ci-dessous :

docker inspect apache_static | grep –i ipaddress

Voici un exemple d'utilisation de la commande pour lancer correctement le container reverse proxy :

docker run -e STATIC_APP=172.17.0.5:80 -e DYNAMIC_APP=172.17.0.6:3000 –p 8080:80 res/apache_rp

Notre configuration:



Étape 6

Pour cette étape nous avons créé une nouvelle branche qui s'appelle «fb-apache-loadbalancing» et qui se base directement sur la branche précédente «fb-dynamic-configuration ».

Nous avons choisi d'implémenter le load balancing dans notre infrastructure. Notre solution fonctionne parfaitement pour la partie statique (le site web) mais ne fonctionne pas pour la partie dynamique (la liste d'emplacements). Nous allons quand même expliquer comment nous avons réalisé cette étape.

Pour le serveur reverse proxy :

Nous avons modifié notre fichier de configuration pour pouvoir utiliser le load balancing

```
<?php
        $dynamic_app_1 = getenv('DYNAMIC_APP_1');
        $dynamic_app_2 = getenv('DYNAMIC_APP_2');
        $static_app_1 = getenv('STATIC_APP_1');
        $static_app_2 = getenv('STATIC_APP_2');
?>
<VirtualHost *:80>
        ServerName demo.res.ch

        <Proxy "balancer://dynamic">
                BalancerMember 'http://<?php print "$dynamic_app_1"?>/'
                BalancerMember 'http://<?php print "$dynamic_app_2"?>/'
        </Proxy>
        ProxyPass '/api/locations/' "balancer://dynamic"
        ProxyPassReverse '/api/locations/' "balancer://dynamic"

        <Proxy "balancer://static">
                BalancerMember 'http://<?php print "$static_app_1"?>/'
                BalancerMember 'http://<?php print "$static_app_2"?>/'
        </Proxy>
        ProxyPass '/' "balancer://static"
        ProxyPassReverse '/' "balancer://static"

</VirtualHost>
```

Pour cela nous avons dû ajouter deux modules dans le DockerFile pour permettre l'utilisation de cette spécificité, les voici : « proxy_balancer lbmethod_byrequests »

Pour vérifier que le load balancing s'effectuait correctement pour la partie statique, nous avons mis un petit script PHP permettant d'afficher l'adresse IP du serveur. À chaque fois qu'on actualise la page l'adresse IP du serveur alterne. On a donc là la vérification que le load balancing s'effectuait correctement.

L'idée pour vérifier la partie dynamique était plus ou moins la même. Nous avons mis l'adresse IP dans le premier élément du tableau des listes d'emplacement. L'idée était tout simplement de l'afficher pour vérifier qu'elle alternait bien elle aussi.

Voici l'erreur que nous avons lorsque nous essayer d'accéder à la ressource http://demo.res.ch:8080/api/locations/

```
« Cannot GET // »
```

Voici un exemple d'utilisation de la commande pour lancer correctement le container reverse proxy :

docker run -e STATIC_APP_1=172.17.0.2:80 -e STATIC_APP_2=172.17.0.2:80 -e DYNAMIC_APP_1=172.17.0.4:3000 -e DYNAMIC_APP_2=172.17.0.5:3000 -p 8080:80 apache_rp

Voici notre configuration pour cette étape: