MACHEREZ Alexis

Année 2018

Rapport de PFE

Application de gestion de conteneurs pour sites web

POLYTECH Lille

Boulevard Paul Langevin

Cité Scientifique

59655 VILLENEUVE D'ASCQ

Tuteurs : Xavier REDON

Thomas VANTROYS

Sommaire

Introduction

Les infrastructures système réseau sont omniprésentes, que ce soit pour héberger des serveur ou pour gérer le réseau interne en entreprise. Même un réseau domestique est une architecture système réseau, la “box” étant un routeur qui fait passerelle vers l’extérieur pour les machines connectées dessus.

C’est infrastructures ont évolué au fil du temps, devenant plus complexe et aussi plus efficace. En effet l’apparition de la virtualisation dans les années 1990 a rendu l’administration système réseau plus efficace et plus complexe tout apportant une plus grande simplicité de mise en place et de mise à jour. Cela permet entre autre d’héberger plusieurs services de manière isolée sur un même serveur physique. Un processeur peut accueillir plusieurs machines virtuelles, ce qui permet alors d’optimiser les ressources “hardware”.

Depuis quelques années, une autre forme de virtualisation/isolation a fait son apparition : les conteneurs. Ces conteneurs permettent d’isoler un ou plusieurs processus sur une machine et bénéficient d’une grande versatilité ainsi que d’une capacité de déploiement encore plus grande que les machines virtuelles.

Les grands hébergeurs tels que Facebook, OVH ou Google utilisent cette technologie à grande échelle notamment pour faire du load balancing, leur permettant d’adapter la capacité d’écoute de leurs serveurs en fonction du nombre de clients de façon aisée.

Ce projet de fin d’étude aborde cette technologie pour de l'hébergement de site web adaptatif.

1. Présentation du projet de fin d'étude

a. Contexte

Disponible depuis de nombreuses années dans le kernel linux, les conteneurs sont de plus en plus utilisés dans le contexte de virtualisation et d'isolation. Ceux ci jouissent d'une grande facilité de déploiement ainsi que d'une grande versatilité. En effet, leur capacité à pouvoir être lancé de manière automatisée permet de rendre les architectures système réseau plus adaptées. Cette technologie s'est donc rapidement développée et tend à remplacer l'utilisation massive de machines virtuelles. Ce projet, intitulé "Application de gestion de conteneurs pour site web", utilisera les conteneurs dans le cadre de l'hébergement de site web, proposant alors une structure modulable en fonction des besoins.

b. Cahier des charges

Ce projet regroupe deux principaux objectifs :

-La mise en place d'une architecture système réseau pouvant acceuillir un grand nombres de conteneurs.

-La création d'une application web permettant de lancer à distance un conteneur hébergeant un serveur web.

L'architecture réalisée doit permettre une gestion automatique des conteneurs (construction, lancement, arrêt et destruction). Elle doit ne consommer qu'une seule adresse IPv4 publique, et rendre les conteneurs accessible en IPv4 ou IPv6 par les protocoles http ou https. Les ressources telles que la mémoire vive, le processeur, l'accès au disque dur et la bande passante internet seront contrôlés et partagés entre les conteneurs. Il faudra aussi mettre en place un moyen d'envoyer des fichiers aux conteneurs.

Il faudra donc mettre en place un réseau privé pour connecter l'ensemble des conteneurs qui communiqueront avec l'extérieur par le biais d'un proxy inverse hébergé par la machine hôte. Les ressources utilisées seront contrôlées par les mécanismes cgroups. L'automatisation du déploiement des conteneurs sera assurée par des scripts shell.

L'application web sera accessible par des utilisateurs habilités, ils pourront y créer un site en choisissant un nom de domaine et le type de serveur hébergé, contrôler l'état de leur(s) site(s), lancer ou stopper un site, mettre à jour le contenu d'un site, et détruire un site.

Il faudra aussi prévoir une page administrateur pouvant contrôler l'ensemble des conteneurs.

2. Architecture système

a. Architecture réalisée

L'architecture système réseau que j'ai réalisée dans le cadre de ce projet consiste en une machine virtuelle faisant office d'hôte et hébergeant un proxy inverse qui redirige les requêtes vers les conteneurs concernés. L'interface réseau de cette machine est donc une interface ethernet eth0 configuré avec l'ip publique disponible.

Pour connecter les conteneurs, j'utilise un réseau privé 10.1.1.0/24. Les conteneurs disposent donc des adresses 10.1.1.1 à 10.1.1.253, chaque conteneur sera connecté à un pont br0 configuré en gateway à l'adresse 10.1.1.254 grâce à une paire d'ethernet virtuelle eth0@vif1. Les conteneurs ne seront pas visible de l'extérieur et communiqueront avec internet via le proxy inverse.

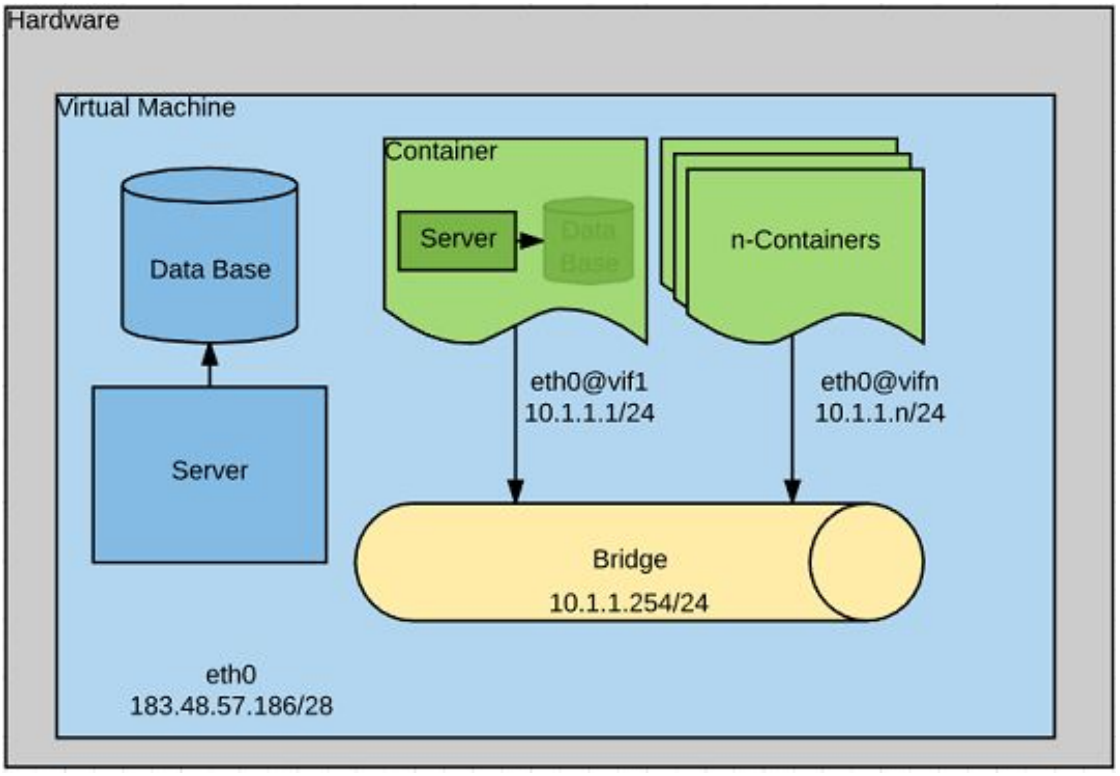


schéma de l'architecture système réseau

b. Préparation de la machine hôte

Pour le début du projet, la machine hôte est une machine virtuelle xen installée sur le serveur de l'école cordouan et mise sur le réseau de travaux pratiques IMA5-SC (193.48.57.0/28).

La machine est configurée par le fichier /etc/xen/ima5-pfe.cfg pour disposer de 512 Mo de RAM, d'un coeur CPU, et de deux disques virtuels. La mise en réseau est assurée par le pont IMA5sc.

Les paquetages apache2, nginx, bind9, debootstrap, unshare, bridge-utils, et ssh ont été installés dans la machine virtuelle.

i. Configuration réseau

La configuration réseau de la machine regroupe une interface ethernet eth0 et un pont br0. La liaison entre les deux interfaces est assurée grâce à des règles iptables. J'ai éditer le fichier /etc/network/interfaces afin de rendre la configuration permanente :

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 static
 address 193.48.57.186
 netmask 255.255.255.240
 gateway 193.48.57.177

auto br0
iface br0 inet static
 address 10.1.1.254
 netmask 255.255.255.0
 bridge_ports none
 bridge_stp off
 bridge_fd 0
```

Les deux interfaces sont configurées avec une ip statique, 193.48.57.186 pour eth0 (193.48.57.177 étant le routeur configuré en protocole réseau avancé) et 10.1.1.254 pour br0. Le pont fait office de gateway pour le réseau privé 10.1.1.0/24 sur lequel les conteneurs seront connectés. Sachant qu'il n'y a pas de conteneur au boot de la machine, le pont ne possède pas de port au lancement, il faut donc mettre la variable bridge_ports à none. Les règles iptables acceptent les ips rentrant sur br0 et les envoient sur eth0 (et inversement).

ii. Serveur de nom

Le serveur de nom (DNS) utilisé sur la machine est bind9, il associe l'adresse ip de la machine à un nom de domaine. J'ai réservé le nom de domaine plille.space sur gandi.net puis configuré bind pour créer la zone plille.space.

Afin de créer la zone, j'ai édité le fichier /etc/bind/named.conf.local :

```
zone "plille.space" {
	type master;
	file "/etc/bind/db.plille.space";
	allow-transfert {217.70.177.40; };
};
```

Configurée ainsi, la zone est le serveur de nom maître (il est possible de créer des zones de type esclave) et est chargée à partir du fichier /etc/bind/db.plille.space configuré de la façon suivante :

```
$TTL    604800
@       IN      SOA     ns1.plille.space. root.plille.space. (
                      2017110901         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800         ; Negative Cache TTL
;
plille.space.   IN      NS      ns1.plille.space.
plille.space.   IN      NS      ns6.gandi.net.
NS1     IN      A       193.48.57.186
NS6     IN      A       217.70.177.40
@       IN      A       193.48.57.186
www     IN      A       193.48.57.186
```

La résolution de la zone plille.space se fait par ns1.plille.space (DNS principal) à 193.48.57.186 ou par ns6.gandi.net (DNS secondaire fournit par gandi) à l'adresse 217.70.177.40. J'ai aussi préciser l'ip de la zone (@) et lier l'alias www à cette zone.

Dans la suite du projet, la création d'un sous-domaine pour un conteneur se fera par l'ajout d'une ligne "sous-domaine IN CNAME plille.space.".

Enfin, j'ai configuré le fichier /etc/bind/named.conf.options :

```
options {
	directory "/var/cache/bind";
	dnssec-validation auto;
	auth-xdomain no;
	listen-on-v6 { any; };
};
```

Il m'a ensuite fallu indiqué cette configuration à gandi. Pour se faire, il faut remplir les champs suivants sur la page web de gestion du domaine :

-Gérer les glues records :

Nom du serveur : ns1.plille.space
IP : 193.48.57.186

-Modifier les serveurs DNS :

DNS1 : ns1.plille.space
DNS2 : ns6.gandi.net

Pour vérifier le chargement de la zone, j'ai utilisé la commande 'named-checkconf -z' avant de redémarrer le serveur de nom avec la commande 'service bind9 restart'. Une fois la zone propagée, j'ai pu accéder aux urls plille.space et www.plille.space sur un navigateur.

iii. Proxy inverse

Le proxy inverse se place en façade des conteneurs, toutes les requêtes lui seront adressées et c'est lui qui va les rediriger vers les conteneurs. Le serveur Nginx est réputé pour bien fonctionner en tant que proxy inverse mais suite à un problème d'installation je me suis rabattu sur un serveur Apache2.

Pour configurer Apache2, j'ai commencé par configurer le nom d'hôte de la machine. Je lui ai donné le nom plille.space avec la commande 'hostname' et l'édition de /etc/hostname. La ligne "193.48.57.186 plille.space" a été ajoutée dans /etc/hosts. Enfin j'ai configuré le fichier /etc/resolv.conf :

```
search plille.space
nameserver 193.48.57.186
```

Les serveurs écouteront sur les ports 80 (http) et 443 (https), je n'ai donc pas eu besoin de changer la configuration des ports d'Apache2.

J'ai d'abord créé un serveur principal qui accueillera par la suite l'application web. Pour cela, il faut créer et éditer un fichier de configuration dans /etc/apache2/sites-available, dans mon cas plille.space.conf :

```
<VirtualHost *:80>
        ServerName plille.space
        ServerAlias www.plille.space
        DocumentRoot /var/www/html/
        ErrorLog /var/log/apache2/error.log
        CustomLog /var/log/apache2/access.log combined
</VirtualHost>
```

Il suffit ensuite d'appliquer la configuration avec 'a2ensite' qui est équivalent à 'ln -s /etc/apache2/sites-available/ /etc/apache2/sites-enabled/' puis de recharger le serveur Apache2 avec 'service apache2 reload'.

Pour mettre en place le proxy inverse, j'ai choisi de créer un virtualhost à la création d'un conteneur qui écoutera sur le sous-domaine associé au conteneur. Chacun de ces virtualhosts aura la forme suivante :

```
<VirtualHost *:80>
        ServerName sous-domaine.plille.space
        ProxyPass / http://10.1.1.x:80/
        ProxyPassReverse / http://10.1.1.x:80/
        ProxyRequests Off
</VirtualHost>
```

Où "x" est représente l'adresse ip d'un conteneur (ex : 10.1.1.1).
ProxyPass et ProxypassReverse indiquent respectivement où rediriger les requetes adressés à sous-domaine.plille.space et où chercher les réponses du domaine sous-domaine.plille.space.

Il suffit enfin d'activer les module proxy_http et proxy d'Apache2 avec la commande 'a2enmod proxy_http'. Ces module autorise Apache2 a fonctionné en proxy inverse et à utiliser les protocole http et https pour la redirection.

iv. Accès HTTPS

Pour permettre l'accès au domaine principale et aux sous-domaines en https, il faut obtenir des certificats SSL pour chaque domaine. Une solution gratuite pour obtenir ces certificats est Letsencrypt. J'ai commencé par installer le programme certbot disponible à l'adresse : `https://dl.eff.org/certbot-auto` . Ce programme permet l'obtention de certificat en se connectant au serveur pour vérifier que l'utilisateur demandant le certificat possède bien ce serveur.

Pour débuter, j'ai généré un certificat pour le domaine principal. Il existe plusieurs options, on peut choisir une installation pour apache, ngninx ou manuelle. J'ai testé la commande suivante, qui est conseillée par Letsencrypt :

```
./certbot-auto --authenticator standalone --installer apache --pre-hook "apachectl -k stop" --post-hook "apachectl -k start" --register-unsafely-without-email
```

Le certificat est bien généré mais le plugin --instaler apache de certbot fait planter le serveur apache qui ne redémarre pas dans la foulé. Ce n'est donc absolument pas une solution viable pour l'obtention des certificats dans le cas du déploiement automatisé d'un conteneur.

Il faut alors utiliser la méthode manuelle. Celle-ci est bien plus efficace et dans ce cas, certbot n'intervient pas sur le serveur, ce qui évite les bugs intempestifs. J'utilise donc la commande :

```
./certbot-auto certonly  -n --register-unsafely-without-email --webroot --webroot-path=
/root/sous-domaine/var/www/html -d sous-domaine.plille.space
```

La commande fonctionne, cependant il faut veiller à ce que le conteneur soit lancé et que le reverse proxy inverse ne redirige pas les requêtes vers https.

Ensuite il suffit de changer manuellement la configuration du proxy inverse :

```
<VirtualHost *:80>
        ServerName sous-domaine.plille.space
        ProxyPass / http://10.1.1.1:80/
        ProxyPassReverse / http://10.1.1.1:80/
        ProxyRequests Off
        RewriteEngine on
        RewriteCond %{SERVER_NAME} =sous-domaine.plille.space
        RewriteRule ^ https://%{SERVER_NAME}%{REQUEST_URI} [END,NE,R=permanent]
</VirtualHost>
<IfModule ssl_module>
        <VirtualHost *:443>
                ServerName sous-domaine.plille.space
                SSLEngine on
                SSLVerifyClient None
                SSLProxyEngine on
                ProxyPass / http://10.1.1.1:80/
                ProxyPassReverse / http://10.1.1.1:80/
                ProxyRequests Off
                SSLCertificateFile /etc/letsencrypt/live/sous-domaine.plille.space/fullchain.pem
                SSLCertificateKeyFile /etc/letsencrypt/live/sous-domaine.plille.space/privkey.pem
                Include /etc/letsencrypt/options-ssl-apache.conf
        </VirtualHost>
</IfModule>
```

Cette configuration redirige les requêtes adressées en http au port80 sur le port 443 en https.

c. Déploiement des conteneurs

Pour déployer un conteneur hébergeant un serveur accessible depuis l'extérieur, il y à plusieurs étapes à réaliser.

Premièrement, il faut installer le système de fichiers sur lequel le conteneur tournera. Pour cela il suffit d'utiliser debootstrap :

```
debootstrap --include "$(cat ./packages.txt)" wheezy ./conteneur
http://ftp.us.debian.org/debian
```

Cela va installer une distribution debian wheezy dans le dossier conteneur, en incluant les paquets contenus dans packages.txt (par exemple : apache2, php5, php5-cli, libapache2-mod-php5).

On peut ensuite configurer l'interface réseau du conteneur. Pour cela il faut modifier le fichier conteneur/etc/network/interfaces :

```
auto eth1
iface eth1 inet static
 address 10.1.1.1
 netmask 255.255.255.0
 gateway 10.1.1.254

 post-up ip route add 10.1.1.254 dev eth1 scope link
 post-up ip route add default via 10.1.1.254 dev eth1"
```

Le conteneur pourra communiquer avec le pont de la machine hôte grâce aux configurations d'ip route.

Il est aussi nécessaire de configurer les hosts et la resolution du dns. Ceci est fait en modifiant conteneur/etc/resolv.conf :

```
search conteneur.plille.space
nameserver 10.1.1.254
```

Et en ajoutant la ligne "10.1.1.1 conteneur.plille.space" dans conteneur/etc/hosts.

L'étape suivante consiste à configurer le serveur sur l'hôte et dans le conteneur et à mettre à jour le serveur de nom.
Sur la machine hôte, il suffit d'ajouter la ligne suivante à /etc/bind/db.plille.space :

```
conteneur       IN      CNAME   plille.space.
```

Le sous domaine est directement disponible par la commande dig, mais la propagation sur les navigateurs prendra plus de temps.

Toujours sur l'hôte, j'ajoute un virtualhost permettant de rediriger les requêtes envoyées à conteneur.plille.space. J'ai donc créé et édité un fichier /etc/apache2/sites-available/conteneur.plille.space.conf :

```
<VirtualHost *:80>
        ServerName conteneur.plille.space
        ProxyPass / http://10.1.1.1:80/
        ProxyPassReverse / http://10.1.1.1:80/
        Proxyrequets Off
</VirtualHost>
```

Puis, de la même manière j'ai configuré le serveur dans le conteneur avec le fichier conteneur/etc/apache2/sites-available/conteneur.plille.space.conf :

```
<VirtualHost 10.1.1.1:80>
        ServerName conteneur.plille.space
        ErrorLog /var/log/apache2/error.log
        CustomLog /var/log/apache2/access.log combined
        DocumentRoot /var/www/html
</VirtualHost>
```

L'hôte et le conteneur étant désormais configurés, on peut lancer le conteneur avec unshare :

```
unshare -p -f -u -n --mount-proc=/proc chroot conteneur) &
```

Puis créer le paire et en envoyé une partie dans le conteneur :

```
ip link add vif1 type veth peer name eth1@vif1
ip link set master br0 dev vif1
ip link set vif1 up
ip link set eth1@vif1 netns /proc/"$PID"/ns/net name eth1
```

Le conteneur est alors opérationnels, il est donc possible de générer le certificat ssl :

```
./certbot-auto certonly  -n --register-unsafely-without-email --webroot --webroot-path=
/root/conteneur/var/www/html -d conteneur.plille.space
```

Pour stopper le conteneur, il suffit de recherche le PID du processus faisant tourner le serveur dans le conteneur puis de le kill. Suite à l'arrêt du serveur, l'espace de nom créé par unshare est détruit.

La destruction d'un conteneur se fait par la suppression du système de fichiers du conteneur, suppression du certificat, suppression du proxy inverse et mise à jour du serveur de nom.

Toutes ces différentes étapes sont réalisées dans des scripts shell pour automatiser le déploiement des conteneurs.

Les différents scripts utilise un fichier texte afin d'avoir accès aux différentes informations (nom, ip, pid). Il est construit de la forme suivante :

```
nom%ip%pid
```

Le script create_cont.sh réalise toutes les étapes de configuration permanentes.
Voici son fonctionnement :

```
sh  create_cont.sh  site1  1
```

- vérifie les arguments
- met à jour le dns pour que le domaine site1.plille.space soit disponible
- installe le système de fichiers et les paquetages avec debootstrap
- met à jour l'annuaire des conteneurs
- configure une première fois le proxy inverse sur l'hôte
- configure l'host et la résolution dans le conteneur site1
- configure le fichier interfaces du conteneur site1
- configure le serveur du conteneur site1
- lance le conteneur site1
- génère le certificat pour le domaine site1.plille.space
- stop le conteneur site1
- configure une deuxième fois le proxy inverse pour l'accès en https

Le script launch_cont.sh lance un conteneur.
Voici son fonctionnement :

```
sh  launch_cont.sh  site1
```

- vérifie les arguments
- récupère l'ip du conteneur dans l'annuaire
- créé le pair vif1@eth1
- monte vif1 sur br0
- lance le conteneur avec unshare
- récupère le PID du conteneur et l'écrit dans l'annuaire
- envoie eth1 dans le conteneur avec netns

Le script inside_cont.sh s'execute dans le conteneur lors du lancement avec launch_cont.sh.
Voici son fonctionnement :

sh inside_cont.sh site1

- attend de recevoir le paire eth1@vif1
- monte eth1
- lance le serveur
- attend un kill

Le script stop_cont.sh arrête un conteneur et supprime le pair.
Voici son fonctionnement :

sh stop_cont.sh site1

- vérifie les argurments
- supprime le PID du unshare dans l'annuaire
- recherche le PID de inside_cont.sh
- tue ce processus, l'espace de nom et vif1@eth1 sont détruits

Le quatrième script (destroy_cont.sh) détruit toutes traces du conteneur.
Voici son fonctionnement :

sh destroy_cont.sh site1

- vérifie les arguments
- met à jour l'annuaire
- supprime le répertoire site1
- supprime le certificat
- supprime le virtualhost (proxy inverse)
- supprime le sous-domaine (dns)

La combinaison de ces cinqs scripts (disponibles en annexe) permet de gérer facilement les conteneurs. J'ai essayer avec plusieurs conteneurs à la fois, tout a fonctionné correctement.

3. Application web

a. Création d'une base de données

Pour ce projet, j'ai utilisé mysql pour gérer la base de donnée.

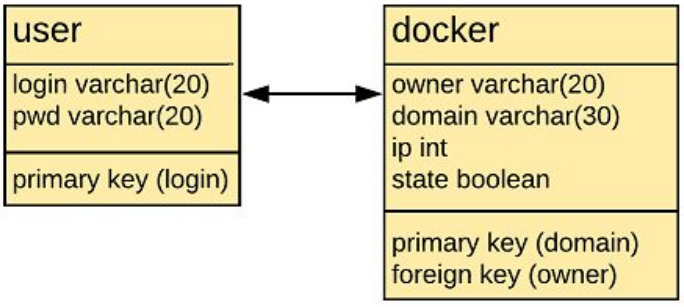


diagramme de la base de données

Mysql dispose d'un shell permettant la gestion des bases. Ce dernier est accessible avec la commande mysql -p.
J'ai commencé par créer la base :

```
create table bdd;
```

Puis j'ai ajouté les tables dans la base (voir diagramme ci-dessus) :

```
use bdd;
create table user (
login varchar(20) primary key,
pwd varchar(20)
);
create table user (
login varchar(20) primary key,
pwd varchar(20)
);
```

Une fois la base créée, on peut ajouter des valeurs dedans :

```
use bdd;
insert into user (login, pwd) values ('login', 'pwd');
insert into docker (owner, domain, ip, state) values ('login', 'domain', 1, 0);
```

Il faut aussi permettre l'accès par apache :

```
use sql;
grant all privileges on *.* to 'root'@'plille.space' identified by 'root' with grant option;
flush privileges;
```

La base est maintenant opérationnelle et accessible depuis l'application web.

b. Utilisation de PHP

PHP est très utile pour coder une application web. Il dispose d'outils permettant la gestion de base données et l'exécution de commande côté serveur.
Un fichier avec l'extension .php permet d'utiliser php en plus d'html (et css) , il suffit dans ce cas d'entourer toutes parties de code PHP par les balises <?php … ?>.

L'envoie de formulaire post html permet à PHP de récupérer des informations d'une page web à l'autre. Il se présente sous la forme suivante :

```
<html>
<p>
<form method=post action=page_suivante.php>
    <input type="text" name="login" id="login">
    <input type="text" name="pwd" id="pwd">
    <input type="submit" class="btn btn-primary">
</p>
</form>
</html>
```

En appuyant sur le bouton de type submit, les champs entrées dans name et pwd seront accessible à la page_suivante.php grâce au code :

```
<?php
$login=$_POST['login'];
$pwd=$_POST['pwd'];
?>
```

Il est aussi possible de faire transiter des données par l'url, il faut ajouter ?var1=val1&var2=val2…
On peut modifier l'url avec html (+php dans ce cas):

```
<html>
href='<?php echo "menu.php?login=";echo $login;echo "&pwd=";echo $pwd ?>'
</html>
```

Ou avec uniquement php :

```
<?php
header("Location: https://plille.space/menu.php?login=$login&pwd=$pwd");
?>
```

Ces informations sont récupérées à la page plille.space/menu.php grâce à :

```
<?php
$login=$_GET['login'];
$pwd=$_GET['pwd'];
?>
```

J'ai beaucoup utilisé ces deux méthodes lors de la réalisation de l'application pour transmettre des données utiles telles que le login de l'utilisateur, le nom d'un conteneur ou certaine variables.

Comme je l'ai indiqué au dessus, PHP permet la gestion de base de données et l'exécution de programme côté serveur.
Par exemple l'application peut se connecter à la base de données pour obtenir, changer, insérer ou supprimer des valeurs :

```
$link=mysql_connect('plille.space','root','root');
mysql_select_db('bdd');
$result=mysql_query("select pwd from user where login='$input'");
$field = mysql_fetch_assoc($result);
echo $field['pwd'];
mysql_query("update docker set state=1 where domain='$input'");
mysql_query("insert into docker (owner, domain, ip ,state) values
('$input','$name','$ip',0)");
mysql_query("delete from docker where domain='$site'");
```

En complément d'html et css, cela permet d'afficher de manière propre les différents sites de l'utilisateurs et les fonctions associés (lancement, arrêt, création…)

D'autre part, la fonction shell_exec de PHP permet d'utiliser mes scripts shell depuis l'application web.
Il m'a fallu commencer par autoriser l'exécution des scripts par apaches. Pour ce faire il faut changer les droits des scripts :

```
chown www-data:root script.sh
```

Et éditer le fichier /etc/sudoers :

```
www-data ALL=(ALL) NOPASSWD:/root/script.sh
```

Il m'a ensuite été possible d'utiliser la commande shell_exec, par exemple :

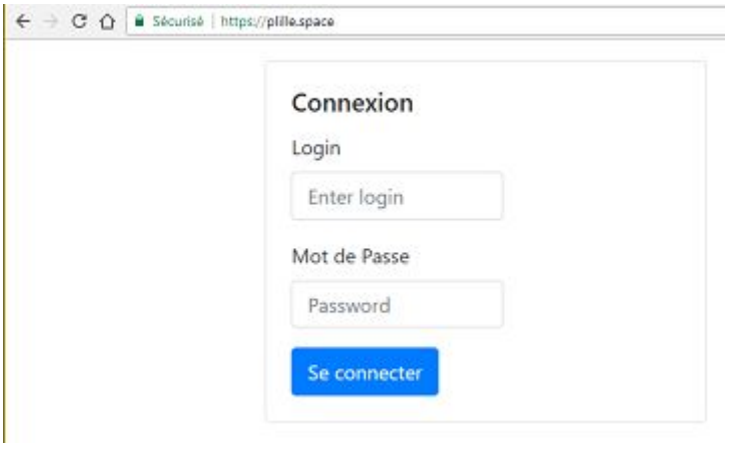
```
shell_exec("sudo /root/create_cont.sh $domain $ip");
```

Il faut tout de même noter qu'apache s'exécute dans /var/www/html, j'ai donc ajouter cd /root au début de tout mes scripts.

c. Présentation de l'application web réalisée

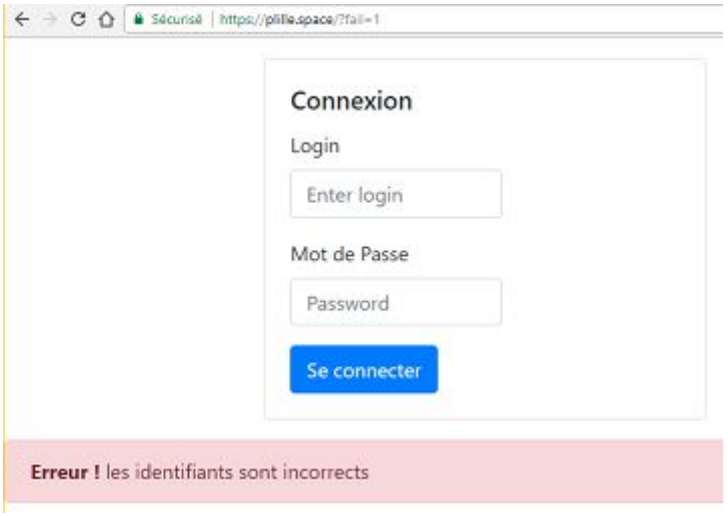
Les codes utilisés pour l'application web sont disponibles dans une archive zip sur le wiki : https://projets-ima.plil.fr/mediawiki/index.php/Fichier:Code_html.zip

i. Connexion



screenshot de la page de connexion

La première page accessible à l'adresse https://plille.space est index.php. Cette page propose à l'utilisateur d'entrer ses identifiants pour se connecter. Le login et le mot de passe sont récupérés sur la page connexion.php (que l'utilisateur ne verra pas) par la méthode post (formulaire) afin d'être vérifiés. S'ils sont valides, l'utilisateur accède à https://plille.space/menu.php. Sinon il reste sur la page de connexion est une alerte est émise pour lui indiquer le problème.



screenshot de la page de connexion avec l'alerte

ii. Menu principal

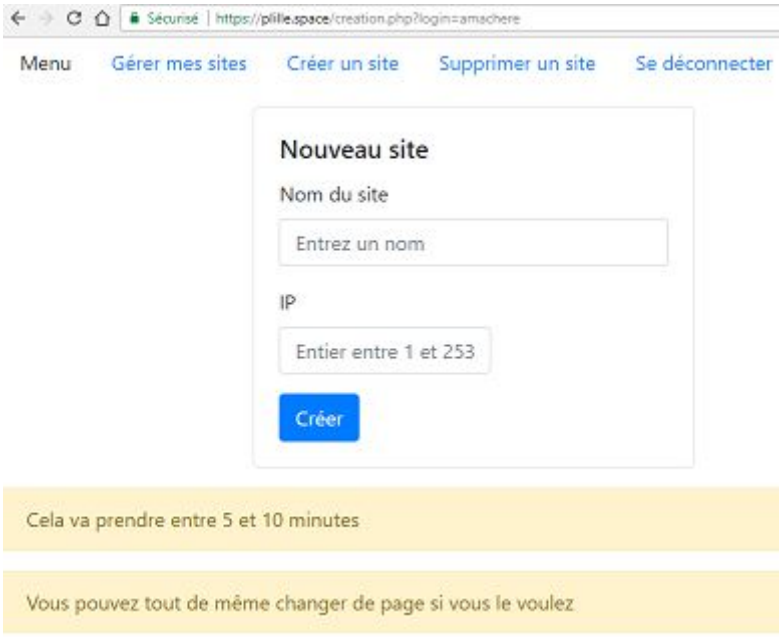
screenshot de la page du menu

Le menu principale permet à l'utilisateur de voir tous les domaines qu'il possède ainsi que leur état (si le serveur tourne ou non). Les domaines sont présentés dans un tableau (une ligne par domaine) avec quatre colonnes. La première colonne est le nom du domaine, la deuxième indique l'état du site. Les colonnes trois et quatres sont des boutons, la troisième permet de lancer ou stopper le serveur de la même ligne, la quatrième est un lien pour accéder à la page de téléchargement de code. A la fin de chaque action (lancer/stopper, télécharger du code, créer/supprimer un site), l'utilisateur est renvoyé sur cette page avec une alerte lui indiquant le bon déroulement de l'action.
Par exemple, après avoir stopper un site :

screenshot de la page du menu avec une alerte

La barre de navigation (en haut de page) est disponible sur toutes les pages de l'application par soucis de praticité.

iii. Créer un site



screenshot de la page de création de site

La page de création de site est accessible en cliquant sur le lien dans la barre de navigation. Elle propose à l'utilisateur d'entrer un nouveau nom de domaine et une ip, ces deux données sont envoyées par formulaire à la page cree.php (non visible par l'utilisateur). Cette dernière page vérifie les champs (disponibilité, validité), s'ils sont corrects il ajoute le nouveau site dans la base de données et créer le site avec shell_exec. Sinon l'utilisateur reste sur la page de création et une alerte est affichée.

iv. Supprimer un site

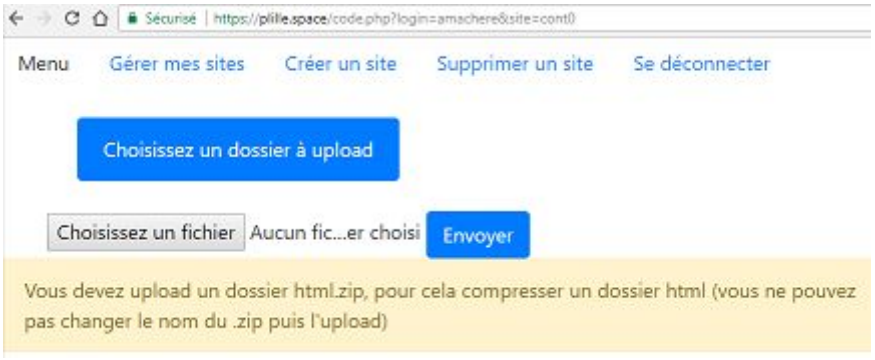
screenshot de la page de suppression de site

La page de suppression est elle aussi accessible par le lien dans la barre de navigation. Sur cette page, l'utilisateur peut cliquer sur le nom d'un de ses domaines. Il est alors envoyé vers validation.php.

screenshot de la page de validation de suppression

La page de validation permet à l'usager de se rétracter en cas d'erreur. S'il clique sur "valider" la page est rechargée avec une variable autorisant php à mettre à jour la base de données et à supprimer le conteneur, puis l'utilisateur est dirigé vers le menu avec une alerte lui indiquant la suppression du site. En cliquant sur "annuler", il est simplement redirigé ver le menu.

v. Télécharger du code



screenshot de la page de téléchargement de code

La page de téléchargement est disponible en cliquant sur le bouton "changer le code" dans le menu. Elle propose à l'utilisateur de télécharger du code dans le conteneur de la ligne du bouton sur lequel il a cliqué. Le fichier doit être un dossier html.zip, il est envoyé à la page upload.php (non visible par l'utilisateur) par le biais d'un formulaire. Le fichier est alors vérifié avant d'être placé sur le serveur puis décompressé dans le conteneur, l'utilisateur est enfin redirigé vers le menu avec une alerte lui indiquant le succès de l'opération. Si le fichier n'est pas valide, la page de téléchargement est rechargée avec une alerte notifiant le problème.

Conclusion

L’application web est fonctionnelle et permet la gestion des conteneurs à distance. Il est possible de créer ou supprimer un site, lancer ou arrêter un conteneur, et de changer le code du site depuis l’application.

D’autre part, ce projet m’a permis d’en apprendre beaucoup sur la programmation orienté back-end dans un premier temps puis front-end dans sa deuxième partie. J’ai pu comprendre davantage comment fonctionne internet que ce soit côté serveur ou côté client.

Annexes

create_cont.sh :

```
#!/bin/bash

echo "$1"%"$2"% >> list_cont.txt

echo "$1       IN      CNAME   plille.space." >> /etc/bind/db.plille.space
service bind9 reload

debootstrap --include "$(cat ./packages.txt)" wheezy ./"$1" http://ftp.us.debian.org/debian

CONF="<VirtualHost *:80>
        ServerName "$1".plille.space
        ProxyPass / http://10.1.1."$2":80/
        ProxyPassReverse / http://10.1.1."$2":80/
        ProxyRequests Off
</VirtualHost>"

touch /etc/apache2/sites-available/"$1".plille.space.conf
echo "$CONF" > /etc/apache2/sites-available/"$1".plille.space.conf
a2ensite "$1".plille.space
service apache2 reload

cp inside_cont.sh "$1"/

echo "127.0.0.1 $1.plille.space" > "$1"/etc/hosts
echo "10.1.1.$2 $1.plille.space" >> "$1"/etc/hosts
echo "$1.plille.space" > "$1"/etc/hostname
echo "search "$1".plille.space" > "$1"/etc/resolv.conf
echo "nameserver 10.1.1.254" >> "$1"/etc/resolv.conf

CONF="
auto eth1
iface eth1 inet static
 address 10.1.1.$2
 netmask 255.255.255.0
 gateway 10.1.1.254

 post-up ip route add 10.1.1.254 dev eth1 scope link
 post-up ip route add default via 10.1.1.254 dev eth1"
echo "$CONF" >> "$1"/etc/network/interfaces

rm "$1"/etc/apache2/sites-available/*
rm "$1"/etc/apache2/sites-enabled/*
CONF="<VirtualHost 10.1.1."$2":80>
        ServerName "$1".plille.space
        DocumentRoot /var/www/html/
</VirtualHost>"
```

```
touch "$1"/etc/apache2/sites-available/"$1".plille.space.conf
echo "$CONF" > "$1"/etc/apache2/sites-available/"$1".plille.space.conf

mkdir -p "$1"/var/www/html
touch "$1"/var/www/html/index.html
echo "$1" > "$1"/var/www/html/index.html

sh launch_cont.sh "$1"
wait 5
./certbot-auto certonly  -n --register-unsafely-without-email --webroot
--webroot-path=/root/"$1"/var/www/html -d "$1".plille.space
sh stop_cont.sh "$1"

CONF="<VirtualHost *:80>
        ServerName "$1".plille.space
        ProxyPass / http://10.1.1."$2":80/
        ProxyPassReverse / http://10.1.1."$2":80/
        ProxyRequests Off
        RewriteEngine on
        RewriteCond %{SERVER_NAME} ="$1".plille.space
        RewriteRule ^ https://%{SERVER_NAME}%{REQUEST_URI} [END,NE,R=permanent]
</VirtualHost>
<IfModule ssl_module>
        <VirtualHost *:443>
                ServerName "$1".plille.space
                SSLEngine on
                SSLVerifyClient None
                SSLProxyEngine on
                ProxyPass / http://10.1.1."$2":80/
                ProxyPassReverse / http://10.1.1."$2":80/
                ProxyRequests Off
                SSLCertificateFile /etc/letsencrypt/live/"$1".plille.space/fullchain.pem
                SSLCertificateKeyFile /etc/letsencrypt/live/"$1".plille.space/privkey.pem
                Include /etc/letsencrypt/options-ssl-apache.conf
        </VirtualHost>
</IfModule>"
echo "$CONF" > /etc/apache2/sites-available/"$1".plille.space.conf
service apache2 reload
echo done
```

launch_cont.sh :

```
#!/bin/bash

IP="$(cat list_cont.txt | grep "$1" | cut -d '%' -f2)"

ip link add vif"$IP" type veth peer name eth1@vif"$IP”

ip link set master br0 dev vif"$IP"

ip link set vif"$IP" up

(nohup unshare -p -f -u -n --mount-proc=/proc chroot "$1" sh inside_cont.sh "$1") &
sleep 0.1
PID="$(ps ax | grep unshare | grep "$1" | grep -v grep | sed 's/^[ ]*//g' | cut -d ' ' -f1)"
sed -i "/$1/s/.*/&$PID/1" list_cont.txt
ip link set eth1@vif"$IP" netns /proc/"$PID"/ns/net name eth1
echo done
```

inside_cont.sh :

```
#!/bin/bash


hostname "$1".plille.space
sleep 1
ifup eth1
a2ensite "$1".plille.space.conf
service apache2 restart
while true
do
        sleep 10
done
```

stop_cont.sh :

```
#!/bin/bash

PID="$(cat list_cont.txt | grep "$1" | cut -d '%' -f3)"
sed -i "s/$PID//1" list_cont.txt
PID="$(ps ax | grep inside | grep "$1" | grep -v unshare | grep -v grep | sed 's/^[ ]*//g' |
cut -d ' ' -f1)"
kill -9 "$PID"
echo done
```

destroy_cont.sh :

```
#!/bin/bash

sed -i "/$1/d" list_cont.txt
rm -rf "$1"
./certbot-auto revoke --cert-path /etc/letsencrypt/live/"$1".plille.space/cert.pem
--non-interactive --quiet
rm /etc/apache2/sites-available/"$1".plille.space.conf
rm /etc/apache2/sites-enabled/"$1".plille.space.conf
service apache2 reload
sed -in "/$1/d" /etc/bind/db.plille.space
service bind9 reload
echo done
```