MACHEREZ Alexis

Année 2017

Rapport de mi-parcours 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..... 3

11. Présentation du projet de fin d'étude

a. Contexte……..……...4

b. Cahier des charges…….…….……...4

12. Travail effectué

a. Architecture réalisée…..……...……..5

b. Préparation de la machine hôte……...……..6

i. Configuration réseau……..6

ii. Serveur de nom…….……...7

iii. Proxy inverse……...8

c. Création et configuration d'un conteneur à la main…..……..10

d. Automatisation du déploiement……...12

13. Tâches prévues……....…...14

a. Terminer le premier objectif…….....14

b. Création de l'application web…….....14

Conclusion.....15

Annexes.....…16

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. Travail effectué

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. Ce même pont communique avec la machine hôte via des règles iptables forward. Les conteneurs ne seront pas visible de l'extérieur et communiqueront avec internet via le proxy inverse.

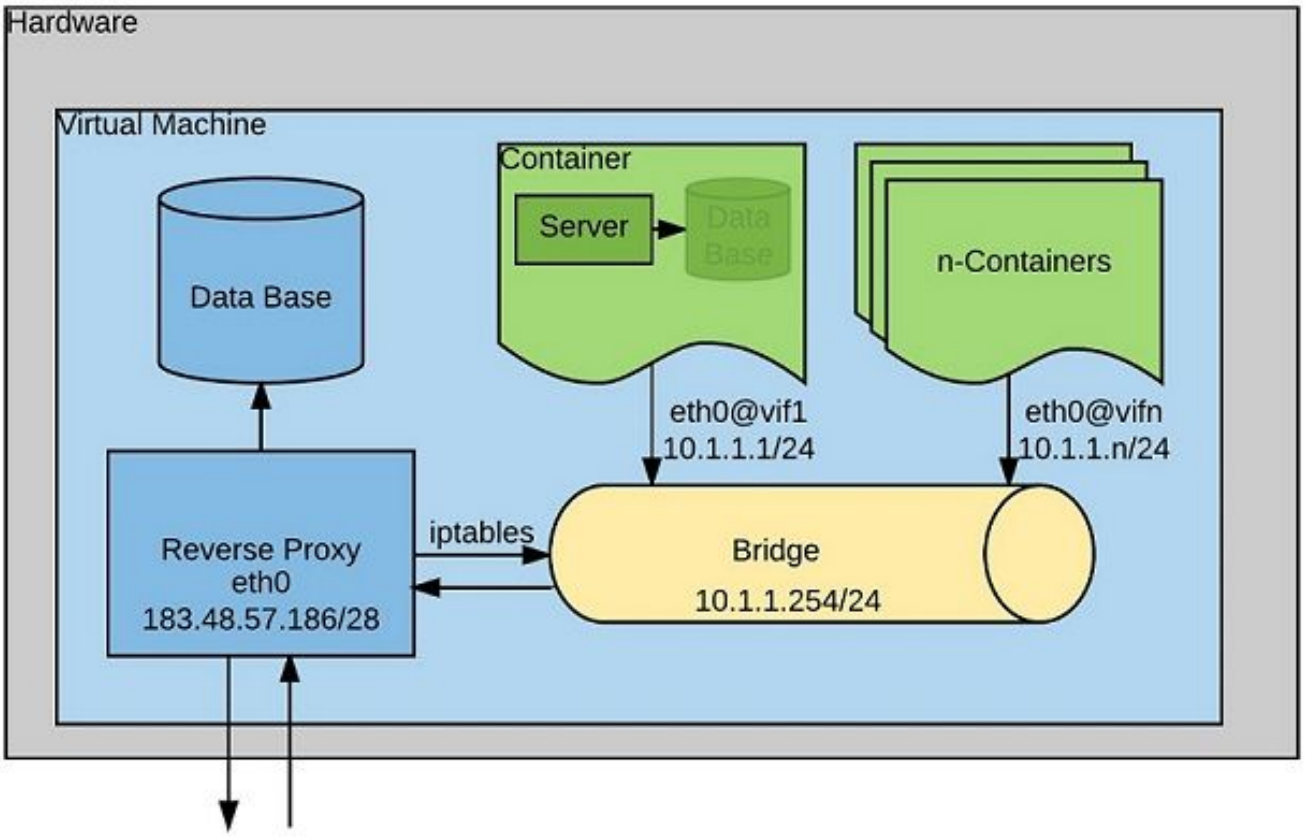

schéma de l'architecture système réseau

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

post-up iptables -I FORWARD -i br0 -o eth0 -j ACCEPT
post-up iptables -I FORWARD -i eth0 -o br0 -m state --state RELATED,ESTABLISHED
-j ACCEPT
```

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). La commande post-up créer les règles après le montage des interfaces réseau lors du lancement et les rend donc permanentes.

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.

c. Création et configuration d'un conteneur à la main

Afin de mieux comprendre l'utilisation et le comportement d'un conteneur, j'ai créer un conteneur à la main hébergeant un serveur écoutant sur un sous domaine de plille.space.

J'ai commencé par installer un système de fichiers sur lequel lancer un conteneur avec la commande :

```
debootstrap --include apache2 wheezy ./conteneurtest http://ftp.us.debian.org/debian
```

Où --include apache2 permet l'installation d'Apache2 dans le système de fichiers, wheezy est la distribution Debian choisie, et conteneurtest est le répertoire accueillant le système de fichiers. L'url est un miroir pour récupérer l'archive Debian.

Une fois l'installation terminée, j'ai lancé le conteneur avec la commande :

```
unshare -p -u -f -n --mount-proc=/proc chroot conteneurtest /bin/bash
```

Qui lance le système de fichiers précédemment installé avec chroot dans un espace de nom.

Je me retrouve donc dans la console du conteneur. Pour continuer les tests, je me suis connecté en ssh sur la machine virtuelle. J'ai ensuite mis le conteneur en réseau grâce à un pair d'ethernet virtuelle :

-depuis l'hôte :

```
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 br0 up
```

Créer le pair et place une extrémité (vif1) sur le pont.

```
ip link set eth1@vif1 netns /proc/"$PID"/ns/net name eth1
```

Envoie l'autre extrémité (eth1) dans le réseau de l'espace de nom du conteneur.

-depuis le conteneur :

```
ip address add dev eth1 10.1.1.1/24
ip link set eth1 up
```

Configure et monte l'interface eth1 du conteneur.

```
ip route add 10.1.1.254 dev eth1 scope link
ip route add default via 10.1.1.254 dev eth1
```

Configure les route empruntée par l'ip du conteneur.

Le conteneur mis en réseau, je peux pinger la machine hôte mais pas l'extérieur. Le proxy inverse permettra d'accéder au site hébergé par le conteneur.

J’ai ensuite créer le sous-domaine conteneurtest.plille.space en ajoutant la ligne “conteneurtest IN CNAME plille.space.” dans /etc/bind/db.plille.space.
On peut accéder à l’url conteneurtest.plille.space, mais c’est la machine hôte qui écoute dessus.

Il m’a donc fallu mettre en place le proxy inverse pour conteneurtest. Comme vu précédemment, j’ai créé un fichier de configuration et rechargé Apache2 :

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

L’url pointe cette fois vers le conteneur, il faut mettre en place le serveur dans le conteneur.

J’ai donc configuré le conteneur de la même manière que la machine hôte. Le nom de domaine du conteneur est conteneurtest.plille.space, la ligne “10.1.1.1 conteneurtest.plille.space” est ajoutée dans /etc/hosts et /etc/resolv.conf est édité :

```
search conteneurtest.plille.space
nameserver 10.1.1.1
```

Puis j’ai configuré le serveur dans le conteneur :

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

L’url affiche maintenant la page web hébergée sur le conteneur.

d. Automatisation du déploiement

L'automatisation est une part importante du projet, pour la réaliser j'ai écrit 4 scripts shell (code en annexes), qui utilise un fichier texte servant d'annuaire. Un cinquième script shell est utilisé par les conteneurs pour se configurer et lancer un serveur.

Le fichier texte contient une ligne par conteneur et se présente comme suit :

NomDuConteneur%IP%PID

La séparation en champs par le caractère "%" est très utile pour extraire une information avec la commande 'cut'.

Le premier script (create_cont.sh) installe le système de fichier, met à jour l'annuaire, configure le proxy inverse, et créer le sous-domaine.
Voici son fonctionnement :

sh create_cont.sh site1 2

- vérifie qu'il y a bien deux arguments
- vérifie que l'argument 2 (index de l'ip) est entier et compris entre 1 et 253
- vérifie si l'ip 10.1.1.2 est libre
- vérifie si le nom site1 est libre
- installe le système de fichiers et les paquetages avec debootstrap
- ajoute la ligne : site1%2% dans l'annuaire
- créer le sous-domaine site1.plille.space
- créer le virtualhost proxy inverse
- place le 5e script dans le répertoire du conteneur

Le deuxième script (launch_cont.sh) lance un conteneur, configure le réseau, et met à jour l'annuaire.
Voici son fonctionnement :

sh launch_cont.sh site1

- vérifie qu'il y a bien un argument
- vérifie si site1 existe
- vérifie si site1 tourne déjà
- récupère l'ip du conteneur dans l'annuaire
- créé le pair vif2@eth1
- monte vif2 sur br0
- lance le conteneur avec unshare, le conteneur lance le 5e script
- récupère le PID du conteneur
- met à jour la ligne : site1%2%498
- envoie eth1 dans le conteneur avec netns
- configure eth1 et les routes dans le conteneur avec nsenter

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

sh stop_cont.sh site1

- vérifie qu'il y a bien un argument
- vérifie si site1 existe
- vérifie si site1 tourne
- récupère le PID du unshare dans l'annuaire
- tue le unshare
- recherche le PID du script exécutant le serveur dans le conteneur
- tue ce processus, le conteneur s'arrête, vif3 est détruit
- met à jour la ligne : site1%2%

Le quatrième script (destroy_cont.sh) détruit le répertoire d'un conteneur, supprime le sous-domaine associé, et supprime le virtualhost proxy inverse.
Voici son fonctionnement :

sh destroy_cont.sh site1

- vérifie qu'il y a bien un argument
- vérifie si site1 existe
- vérifie si site1 est en train de tourner
- supprime la ligne : site1%1% de l'annuaire
- supprime le répertoire site1
- supprime le virtualhost
- supprime le sous-domaine

La combinaison de ces quatres scripts permet de gérer facilement les conteneurs. J'ai essayer avec plusieurs conteneurs à la fois, tout a fonctionné correctement.

3. Tâches prévues

a. Terminer le premier objectif

L'architecture système réseau n'est pas encore tout à fait terminée. Il faut encore que je mette en place les cgroups et leur gestion automatique pour contrôler les ressources. Je dois aussi configuré l'accès en IPv6 et le protocole https, sachant que le proxy inverse d'Apache2 peut supporter https. Enfin, je dois mettre en place un protocole sftp pour échanger des fichiers avec les conteneurs.

L'architecture devra aussi probablement être migrée sur un autre serveur.

b. Création de l'application web

Pour terminer le projet, je vais devoir créer l'application web qui sera utilisée par les utilisateurs. L'annuaire des conteneurs deviendra sûrement une base de données plutôt qu'un fichier texte, cette base de donnée permettra une meilleur gestion des conteneurs à partir de l'application web.

Le serveur de connexion utilisé sera celui de polytech, je n'ai donc pas besoin de gérer de base de données des utilisateurs habilités à créer un site.

Il faut aussi penser à un moyen de mettre à jour le contenu hébergé par un conteneur depuis l'application web.

Conclusion

Ce projet, durant cette première période, m'a permis de me familiariser avec le système et le réseau, des domaines dans lesquels j'avais des lacunes mais qui m'interessent. Même si je me suis heurté à des difficultés pendant ce projet, j'ai réussi à apprendre et à appréhender des techniques que je ne connaissait pas initialement.

La deuxième partie de ce projet sera, je pense, tout aussi intéressante et enrichissante. J'ai en effet fait peu de développement web durant ma formation, ce sera donc l'occasion d'en apprendre plus sur cette discipline.

Annexes

create_cont.sh :

```
#!/bin/bash

if [ "$#" -ne 2 ]; then
	echo "error: Must have two arguments"
	exit
fi

if [ "$2" -gt 253 ] || [ "$2" -lt 1 ]; then
	echo "error: 2nd argument must be integer between 1 and 253"
	exit
fi

cat list_cont.txt | while read ligne
do
	TEST="$(echo $ligne | cut -d '%' -f2)"
	if [ "$2" = "$TEST" ]; then
		echo "error: IP 10.1.1.$2 is already used"
		exit 0
	fi
	TEST="$(echo $ligne | cut -d '%' -f1)"
	if [ "$1" = "$TEST" ]; then
		echo "error: "$1" already exists"
		exit 0
	fi
done
if [ "$?" = "0" ]; then
	exit
fi

debootstrap --include "$(cat ./packages.txt)" wheezy ./"$1" http://ftp.us.debian.org/debian/
echo "$1"%"$2"% >> list_cont.txt
echo "$1	IN	CNAME	plille.space." >> /etc/bind/db.plille.space
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.conf
service apache2 reload
service bind9 restart
cp inside_cont.sh "$1"/
echo done
```

launch_cont.sh :

```
#!/bin/bash

if [ "$#" -ne 1 ]; then
    echo "error : Must have one argument"
    exit
fi

TEST="$(cat list_cont.txt | grep "$1" | cut -d '%' -f1)"
if [ "$1" != "$TEST" ]; then
        echo "error: "$1" doesn't exists"
        exit
fi

TEST="$(cat list_cont.txt | grep "$1" | cut -d '%' -f3)"
if [ -n "$TEST" ]; then
        echo "error: "$1" is already running"
        exit
fi

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
ip link set br0 up
(nohup unshare -p -f -u -n --mount-proc=/proc chroot "$1" sh inside_cont.sh "$1" "$IP") &
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
nsenter -t "$PID" -n ip address add dev eth1 10.1.1."$IP"/24
nsenter -t "$PID" -n ip link set eth1 up
nsenter -t "$PID" -n ip route add 10.1.1.254 dev eth1 scope link
nsenter -t "$PID" -n ip route add default via 10.1.1.254 dev eth1
echo done
```

stop_cont.sh :

```
#!/bin/bash

if [ "$#" -ne 1 ]; then
    echo "error : Must have one argument"
    exit
fi

TEST="$(cat list_cont.txt | grep "$1" | cut -d '%' -f1)"
if [ "$1" != "$TEST" ]; then
        echo "error: "$1" doesn't exists"
        exit
fi

TEST="$(cat list_cont.txt | grep "$1" | cut -d '%' -f3)"
if [ -z "$TEST" ]; then
        echo "error: "$1" is not running"
        exit
fi


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

destroy_cont.sh :

```
#!/bin/bash

if [ "$#" -ne 1 ]; then
    echo "error : Must have one argument"
    exit
fi

TEST="$(cat list_cont.txt | grep "$1" | cut -d '%' -f1)"
if [ "$1" != "$TEST" ]; then
        echo "error: "$1" doesn't exists"
        exit
fi

TEST="$(cat list_cont.txt | grep "$1" | cut -d '%' -f3)"
if [ -n "$TEST" ]; then
        echo "error: "$1" is running"
        exit
fi

sed -in "/$1/d" list_cont.txt
rm -rf "$1"rm /etc/apache2/sites-available/"$1".plille.space.conf
rm /etc/apache2/sites-enabled/"$1".plille.space.confservice
apache2 reload
sed -in "/$1/d" /etc/bind/db.plille.space
service bind9 restart
echo done
```

inside_cont.sh :

```
#!/bin/bash

echo 10.1.1."$2" "$1".plille.space > /etc/hosts
echo "$1".plille.space > /etc/hostname
echo search "$1".plille.space > /etc/resolv.conf
echo nameserver 10.1.1."$2" >> /etc/resolv.con
hostname "$1".plille.space
rm /etc/apache2/sites-available/*
rm /etc/apache2/sites-enabled/*
touch /etc/apache2/sites-available/"$1".plille.space.conf
CONF="<VirtualHost 10.1.1."$2":80>
        ServerName "$1".plille.space
        ErrorLog /var/log/apache2/error.log
        CustomLog /var/log/apache2/access.log combined
        DocumentRoot /var/www/html
</VirtualHost>"
echo "$CONF" > /etc/apache2/sites-available/"$1".plille.space.conf
a2ensite "$1".plille.space.confmkdir -p /var/www/html
touch /var/www/html/index.html
echo "$1" > /var/www/html/index.html
service apache2 restart
while true
do
        sleep 10
done
```