Master 2 SeCReTS

TP Virtualisation et Conteneurs
Deuxième Partie

Première partie
Hyperviseurs

1 VirtualBox (facultatif)

Dans cette partie nous travaillerons sur la machine virtuelle `CLIENT1`.

VirtualBox est un hyperviseur de type 2 très répandu qui possède l'avantage d'être disponible à la fois sous Linux et sous Windows. Dans les TPs précédents nous l'avions installé sur notre machine hôte (Linux ou Windows).

Nous allons dans cette partie faire un bref tour des fonctionnalités et des possibilités qu'offre cette solution de virtualisation et s'approprier les commandes à notre disposition afin de proposer une solution de surf déporté non rémanente.

Si vôtre *host* est une machine Linux, vous pourrez travailler directement sur votre portable. Sinon, commencez par installer cet outil dans la VM (`client1`).

```
# apt-get install VirtualBox
```

Une image ISO regroupant l'ensemble des packages nécessaires est disponible et permet de faire le TP sans accès internet. Il faudra ensuite installer les packages présents dans le répertoire vbox (`dpkg -i vbox/*.deb`). N'oubliez pas après l'installation d'ajouter le compte `user` au groupe `vboxusers` via la commande suivante :

```
# usermod -aG vboxusers user
```

Transfèrerez ensuite l'image `xubuntu` (format *OVA*) fournie vers `client1`.

Grâce à l'utilitaire `VBoxManage`, vous exécuterez les actions suivantes :

- Import de l'image OVA sous le nom `ubuntu`
- Démarrage / Arrêt de `ubuntu`
- Création d'un réseau privé interne
- Connexion de l'image sur le réseau interne
- Création d'un instantané
- Restauration d'un instantané

Après vous êtes familiarisé avec les diverses commandes, vous réaliserez un script shell permettant quelque soit l'état de la machine virtuelle `ubuntu` de la restaurer instantanément dans un état donné (par exemple session ouverte, éditeur de texte démarré) puis un second script permettant de mettre à jour l'instantané (`apt-get dist-upgrade`).

2 Libvirt

Dans cette partie nous travaillerons sur la machine virtuelle `GATEWAY1`.

Libvirt est une autre solution de virtualisation plus orientée vers un usage serveur. Dans cette partie nous transformerons la machine `gateway1` en hyperviseur. Un client graphique sur une autre machine (par exemple `client1` ou `gateway2`) nous permettra d'administrer les machines virtuelles.

2.1 Installation de l'hyperviseur

Installez libvirt sur `gateway1` avec apt (`apt-get install libvirt-bin qemu-kvm`) ou via les packages présents dans le répertoire libvirt (`dpkg -i libvirt/*.deb`).

Après vous être ajouté dans le groupe libvirtd (`usermod -aG libvirtd user`), verifiez que les lignes suivantes sont bien décommentées dans le fichier `/etc/libvirt/libvirtd.conf` :

```
listen_tls = 0
listen_tcp = 1
```

Rajouter la ligne suivante dans le fichier `/etc/default/libvirtd` :

```
libvirtd_opts="-l"
```

Enfin redémarrez le service libvirtd (`systemctl restart libvirtd`). A ce state vous venez de transformer la machine `gateway1` en hyperviseur. Sur quel port écoute le démon libvirtd ?

2.2 Client graphique

Installez sur `client1` le client graphique (`apt-get install virt-manager ssh-askpass qemu-kvm` ou `dpkg -i manager-virt/*.deb`) et connectez vous à l'hyperviseur `gateway1` depuis `client1` en utilisant l'URL suivante : `qemu+ssh://user@gateway1/system`.

Créez une bi-clefs ssh sur `client1` (`ssh-keygen`) et déployez la partie publique sur `gateway1` en utilisant la commande `ssh-copy-id`.

Vous aurez besoin de créer un espace de stockage pour les disques des machines virtuelles (dans les préférences de virt-manager), puis vous pourrez créer une nouvelle VM.

En vous connectant à `gateway1`, validez que la machine virtuelle est bien présente (commande `virsh list --all`) et familiarisez vous avec les différentes options de `virsh`.

```
# Affichage des informations de la VM
$ virsh dumpxml ubuntu
# Démarrage de la VM
$ virsh start ubuntu
# Création d'un snapshot
$ virsh snapshot create-as –name test ubuntu
# Restauration d'un snapshot
$ virsh snapshot revert ubuntu test
```

3 Pour les plus rapides

Après avoir décompressé l'archive `xubuntu.ova` avec l'outil adéquate (la commande `file` est votre amie), convertissez le disque dur de la VM au format `qcow2` à l'aide de la commande `qemu-img` et importez la VM dans libvirt. Vous pourrez ensuite refaire les scripts de la partie sur VirtualBox pour cette VM.

Deuxième partie

Manipulation de conteneurs

Nous allons dans cette partie du TP manipuler les cgroups et namespaces, puis dans un second temps, des conteneurs à l'aide de deux outils : Docker et LXC.

Nous travaillerons sur la machine virtuelle `gateway1`.

1 Cgroups

1.1 Vérification de l'environnement

Commencez par rajouter un CPU à la machine `gateway1`.

Vérifier la présence des commandes de manipulation des cgroups :

```
% which cgcreate
/usr/bin/cgcreate
```

En cas d'erreur, installer le package manquant :

```
# apt install cgroup-tools
```

Vérifier la bonne activation des cgroups :

```
% ls /sys/fs/cgroup
blkio  cgmanager  cpu  cpuacct  cpu,cpuacct  cpuset [...]
```

En cas d'erreur, c'est généralement signe que la distribution ne supporte pas les cgroups. Il est quand même possible de faire apparaître les cgroups avec la commande suivante :

```
# mount -t cgroup cgroup /sys/fs/cgroup
```

1.2 Création d'un nouveau "slice" pour l'utilisateur

Les cgroups sont généralement manipulés par l'utilisateur root. On peut l'observer avec les permissions sur les fichiers dans /sys/fs/cgroup. Mais il est possible de créer des cgroups à destination des utilisateurs.

-> Créez un nouveau cgroup (aussi appelé slice) nommé "test" contrôlable par un utilisateur normal avec la commande cgcreate (indice : utilisez l'option -a). Ce groupe doit contrôler les unités "cpu", "cpuset" et "memory".

Ensuite on doit normalement créer un sous-cgroup dans lequel l'utilisateur pourra effectivement placer des processus. Si vous avez appelé "test" le slice de la question précédente, appelez ce sous-slice "test/1", avec les mêmes unités que précédemment.

-> Créez le slice "test/1".

Enfin vous pouvez effacer ces nouveaux groupes avec la commande cgdelete, ou en faisant un rmdir sur le dossier correspondant dans l'arborescence /sys/fs/cgroup.

1.3 Exécution de commandes dans le nouveau slice

1.3.1 Programme de test

Les nouveaux programmes peuvent être placés dans un slice avec la commande cgexec. Pour disposer de commandes intéressantes à lancer pour tester les cgroups, nous allons installer un outil de benchmark cpu appelé "stress-ng".

```
# apt install stress-ng
```

Ensuite la commande de base à lancer dépend du nombre de CPUs sur votre machine.

```
$ stress-ng --cpu <nb_cpu> -t <nb_secondes>
```

Puis on ajoutera des options pour lancer des tests suivants le cgroup à utiliser.

1.3.2 cpuset

Observez la documentation (man) de cgexec pour voir comment lancer une commande dans votre nouveau cgroup. On lancera le benchmark avec l'option supplémentaire –cpu-method matrixprod pour tester l'unité qui gère l'affectation des processeurs aux programmes en cours d'exécution (cpuset).

Pour voir les paramètres gérés par une unités :

```
% ls /sys/fs/cgroup/<unit>
```

Une fois que vous avez lancé le benchmark, observer avec la commande top ou htop qu'il s'exécute sur tous les processeurs. Si vous n'avez qu'un seul processeur, le test est moins intéressant. Recréez votre machine virtuelle avec plusieurs CPUs, ou passez à la suite.

Pour changer les processeurs affectés à votre cgroup, utilisez la commande cgset :

```
% cgset -r cpuset.cpus=<liste_cpus> test/1
```

Vérifiez avec top ou htop que cela change les processeurs utilisés par le banchmark.

2 Namespaces

Les namespaces de Linux sont généralement complexes à manipuler en ligne de commande. Nous allons voir quelques tests possibles avec la commande "unshare".

Observez la documentation de la commande unshare, et démarrez un nouveau processus bash dans un namespace UTS séparé. Avec la commande hostname, changez le nom de machine ("hostname") dans l'environnement confiné, et observez que cela ne change pas le hostname en dehors de ce nouveau processus. Utilisez ensuite les namespaces PID et Network et mettez en évidence les restrictions sur ces deux types de ressources.

Avec la commande ifconfig ou ip, observez les interfaces visibles par ce nouveau shell.

Remarque : il est également possible de créer un nouveau namespace réseau à l'aide de la commande ip et de placer une de nos interface réseau dans ce nouveau namespace.

3 Docker

3.1 Installation et manipulations classiques

Vous commencerez par installer Docker sur `client1` avec apt (`apt-get install docker.io`) ou via les packages fournis (`dpkg -i deb-docker/*.deb`).

Nous nous baserons sur une image ubuntu. Pour l'obtenir avec un accès internet :

```
$ docker pull ubuntu
```

Vous pouvez également l'importer à partir de l'archive tar :

```
$ cat ubuntu.tar | docker load
```

Via le client `docker`, vous exécuterez les actions suivantes :

- Lister les images installées
- Lancement d'un conteneur ubuntu et affichage de la version du noyau

– Lancement d'un conteneur ubuntu en mode démon, faisant tourner `bash`, ayant pour nom `debian`, affichage des processus courants, arrêt et suppression du conteneur
– Lancement d'un conteneur ubuntu, installation de Python et affichage des différences
– Lancement d'un conteneur ubuntu en mode démon, écoutant sur le port 9090 du *guest*, et affichant le contenu de `/etc/passwd` de l'host lors d'une connexion sur ce port

3.2 DockerFile (facultatif)

Les *Dockerfile* sont un mécanismes permettant de décrire la création d'une image de conteneur. Leurs utilisations est décrite sur le site officiel de Docker.

→ Vous créerez et testerez deux Dockerfile :

– Un premier permettant d'obtenir une image faisant directement la dernière étape de la section Manipulation ;
– Un second permettant d'obtenir une image ubuntu à jour, avec Python installé

4 Installation de LXC

Dans cette partie nous travaillerons sur la machine virtuelle `GATEWAY1`.

Pour ceux qui ne disposent pas d'un accès internet, décompresser l'archive lxc-ubuntu-2018.tgz dans le repertoire `/var/cache/lxc` de `gateway1` (`tar xzvf /root/lxc-ubuntu-2018.tgz -C /var/cache/lxc/`).

Ensuite, vous allez créer un conteneur ubuntu minimal avec la commande suivante :

```
$ lxc-create -n ubuntu1 -t ubuntu
```

Attachez vous à ce conteneur après l'avoir démarré et mettez en évidence le fait que vous êtes dans un conteneur.

Créez un bridge sur lequel sera connecté votre conteneur `ubuntu1`.

```
# brctl addbr lxc_network
# ip link set lxc_network up
# ip addr add 10.1.4.254/24 dev lxc_network
```

La configuration du réseau peut se faire directement au niveau de lxc :

```
# /var/lib/lxc/ubuntu1/config
lxc.net.0.type = veth
lxc.net.0.link = lxc_network
lxc.net.0.flags = up
lxc.net.0.ipv4.address = 10.1.4.1/24
lxc.net.0.ipv4.gateway = 10.1.4.254
```

De la même façon que dans la partie sur docker, lancer dans le conteneur `ubuntu1` un script écoutant sur le port 9090 et affichant le contenu de /etc/passwd lors d'une connexion sur ce port.