KVM, une technique de virtualisation simple et efficace

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 9 mars 2008. Dernière mise à jour le 16 avril 2008

https://www.bortzmeyer.org/kvm.html

Dans la grande famille des solutions de virtualisation, permettant de faire tourner un système d'exploitation sur un autre, KVM, spécifique à Linux, permet d'exécuter du code natif, donc avec de bonnes performances, mais sans nécessiter d'aide de la part du système d'exploitation hébergé (donc on peut lancer un autre système que Linux).

KVM est un module du noyau Linux qui offre un certain nombre de services de virtualisation permettant à sa partie en mode utiliseur, l'émulateur Qemu <https://www.bortzmeyer.org/qemu.html> de parler au matériel avec efficacité. Notamment, si le processeur de la machine **hôte** est le même que celui de la machine **hébergée**, il n'y a pas besoin d'une phase d'interprétation, le code peut être évalué directement, donc à vitesse normale (Qemu seul est très lent).

Le fait que KVM+Qemu présentent une véritable machine virtuelle permet de faire tourner des systèmes d'exploitation non modifiés, dont on n'a qu'un binaire, comme par exemple Microsoft Windows. Si le système hôte doit être un Linux, le système hébergé est libre, contrairement à User-Mode-Linux <https://www.bortzmeyer.org/user-mode-linux.html>. Contrairement à Xen <https://www.bortzmeyer.org/xen.html>, il ne nécessite pas de modifier le système d'exploitation invité. Je fais ainsi tourner un FreeBSD sans aucune modification sur un PC Ubuntu Linux.

KVM ne fonctionne pas entièrement en logiciel. Il a besoin d'un support du processeur (qui est légèrement différent entre Intel et AMD, toutes les instructions ultérieures sont pour un Intel). Sur mon Dell Latitude D430, ce support n'est pas activé automatiquement, il faut le faire dans le BIOS. Même chose sur les gros serveurs Dell Poweredge, il faut activer l'option "*Virtualization*" dans les options CPU. On peut ensuite tester qu'il est bien activé avec :

```
% grep vmx /proc/cpuinfo
```

Prenons maintenant l'exemple de l'installation de FreeBSD 7. Je télécharge une image ISO. On doit d'abord créer le fichier qui servira de disque dur à la machine virtuelle :

```
% qemu-img create -f qcow Machines-Virtuelles/snape-freebsd.qemu-cow 1400M
```

Ensuite, je lance KVM avec le script suivant (pour charger les modules noyau nécessaires) :

```
#!/bin/sh

cd $HOME/Machines-Virtuelles

sudo modprobe kvm
sudo modprobe kvm-intel

kvm -m 512 -hda snape-freebsd.qemu-cow -cdrom $HOME/tmp/7.0-RELEASE-i386-bootonly.iso -boot d
```

(Les options de la commande `kvm` sont les mêmes que celles de `qemu`.) Le programme d'installation de FreeBSD se lance alors et on peut installer FreeBSD comme d'habitude. Une fois que cela est fait, on peut désormais lancer KVM sur le « disque dur » sans CDROM :

```
% kvm -m 512 -hda snape-freebsd.qemu-cow -boot c
```

À noter que, si on ne veut pas installer soi-même, on trouve énormément d'images toutes prêtes, pour les principaux systèmes libres, sur Free OS Zoo <http://free.oszoo.org/>.

Pour le réseau, c'est un peu plus compliqué. Il existe de nombreuses méthodes (décrites dans la documentation de Qemu <http://qemu.org/user-doc.html>). J'utilise ici `tap`, qui malheureusement nécessite d'être root pour lancer `kvm`, mais fournit le plus de possibilités. Le script est donc modifié ainsi :

```
sudo kvm -m 512 -hda snape-freebsd.qemu-cow -net nic -net tap -boot c
```

KVM lancera `/etc/kvm/kvm-ifup` qui, chez moi, contient :

```
#!/bin/sh -x

# My code:

iface=$1

# The Qemu host
myipv4addr=$(ip addr ls  eth0 | awk '/inet / {print $2}' | cut -d/ -f1)

remipv4addr=10.1.1.42

# IPv4
sudo /sbin/ifconfig $iface $myipv4addr  netmask 255.255.255.255
sudo route add -host $remipv4addr dev $iface

# activate ip forwarding
sudo sh -c 'echo 1 > /proc/sys/net/ipv4/ip_forward'

# activate ARP proxy to "spoof" arp address
sudo sh -c "echo 1 > /proc/sys/net/ipv4/conf/eth0/proxy_arp"
sudo sh -c "echo 1 > /proc/sys/net/ipv4/conf/${iface}/proxy_arp"

# set "spoofed" arp address
sudo arp -Ds $remipv4addr eth0 pub
```

Si on veut gérer plusieurs machines virtuelles, chacune avec sa propre adresse, une solution possibe est de modifier le début du script ainsi :

```
if [ $iface == "tap0" ]; then
    remipv4addr=10.1.82.3
elif [ $iface == "tap1" ]; then
    remipv4addr=10.1.82.4
else
    exit 1
fi
```

Les numéros des interfaces `tap` seront attribués dans l'ordre de démarrage. Si on veut éviter que les machines ne changent d'adresse IP selon cet ordre, on peut définir un numéro explicitement en remplaçant `-net tap` par `-net tap,ifname=tapN`. (Il serait bien agréable que la valeur de l'option `-name` soit passée au script `kvm-ifup` mais cela ne semble pas être le cas.)

Et pour IPv6 ? C'est plus complexe car l'équivalent du "*proxy ARP*" est très pénible :

```
myipv6addr=$(ip addr ls  ${lan_iface}  | awk '/inet6 / {print $2}' | \
       cut -d/ -f1 |  head -n1)

if [ $iface == "tap0" ]; then
    ...
    remipv6addr=2001:660:3003:3::1:3
...

# IPv6
sudo /sbin/ifconfig $iface add $myipv6addr
sudo route add -Ainet6 $remipv6addr dev $iface

# Proxy NDP (Neighbor Discovery Protocol)
sudo ip -6 neigh add proxy $remipv6addr dev ${lan_iface}

# Apparently, you need to put *every* IPv6 address of the LAN here :-(
# http://gentoo-wiki.com/IPV6_And_Freebox
lanipv6addr="2001:660:3003:3::1 2001:660:3003:3::1:1 2001:660:3003:3::1:2 2001:660
:3003:3:219:b9ff:fee4:25f9"
for addr in $lanipv6addr; do
    sudo ip -6 neigh add proxy $addr dev $iface
done
```

On notera que la console par défaut de KVM ne permet pas le copier/coller, ce qui est très pénible si on veut signaler une bogue `<http://kvm.qumranet.com/kvmwiki/Bugs>` en donnant tous les détails. La méthode que j'utilise alors est la console série. On ajoute l'option `-serial file:tmp/MAMACHINE.log` au lancement de KVM et, aux options de démarrage du noyau Linux (si on utilise Grub, ces options peuvent se changer en tapant 'e' comme "*Edit*"), `console=ttyS0,115200`.