Guides d'installation

- Installer un hôte de virtualisation QEMU/KVM (Fedora Server 31)
- Création d'un bridge pour QEMU/KVM avec NetworkManager et Libvirt

Installer un hôte de virtualisation QEMU/KVM (Fedora Server 31)

fedora27server-945x400.jpg

Fonctionnement de QEMU/KVM

KVM (Kernel-based Virtual Machine) est une technologie de virtualisation Open Source intégrée à Linux. Avec KVM, vous pouvez transformer Linux en un hyperviseur qui permet à une machine hôte d'exécuter plusieurs environnements virtuels isolés, appelés invités ou machines virtuelles.

KVM convertit le noyau Linux en un hyperviseur de type 1. Pour exécuter des machines virtuelles, tous les hyperviseurs ont besoin de certains composants au niveau du système d'exploitation : gestionnaire de mémoire, planificateur de processus, pile d'entrées/sorties (E/S), pilotes de périphériques, gestionnaire de la sécurité, pile réseau, etc. La technologie KVM comprend tous ces composants, car elle est intégrée au noyau Linux. Chaque machine virtuelle est mise en œuvre en tant que processus Linux standard. Elle est gérée par le planificateur Linux standard et dispose de matériel de virtualisation dédié (carte réseau, carte graphique, un ou plusieurs processeurs, mémoire, disques).

400px-Kernel-based_Virtual_Machine.png

Installation d'un hôte de virtualisation QEMU/KVM (avec Libvirt)

- **Commençons par le commencement, les détails de mon installation Fedora Server :**

Partition	Taille	Système de fichiers
/boot/efi	512 MiB	EFI Partition
/	10 GiB	XFS
/datastore	[*le reste*]	XFS

- **Installons ensuite les paquets nécessaires à l'installation d'un hôte de virtualisation QEMU-KVM :**

- **qemu-kvm :** package de base pour installer QEMU/KVM et les dépendances requises.
- **libvirt :** l'API de management (utilisée par Virt-Manager)
- **libvirt-python :** des bindings Python utiles pour automatiser
- **libguestfs-tools-c** : un ensemble d'outils pour manipuler des images de VM (version sans dépendances Perl)
- **virt-install :** permet d'installer les VM en ligne de commandes
- **virt-v2v :** permet de pouvoir convertir les VM dans des formats différents
- **tuned :** un outil qui tweak des paramètres systèmes en fonction de profils
- **cockpit (+ cockpit-machines)** : une interface Web de gestion, capable de gérer les VM, le stockage, le réseau...etc. (optionnel)

```
# Installer les packages nécessaires
dnf install -y qemu-kvm libvirt python-libvirt libguestfs-tools-c virt-install virt-v2v virt-top tuned cockpit cockpit-machines

# Supprimer le dashboard (inutile pour un hôte de virtualisation)
dnf remove -y cockpit-dashboard
```

- **Activons le service "libvirtd" :**

```
systemctl enable --now libvirtd
```

- **Par défaut, Libvirt crée un réseau NAT. Ce dernier n'est peut-être pas utile selon les configurations. Il est possible de le supprimer** :

```
virsh net-destroy default

virsh net-undefine default

systemctl restart libvirtd
```

- **Si vous utilisez Cockpit, quelques étapes sont requises avant de pouvoir l'utiliser** :

```
# Ajouter un règle dans le firewall pour autoriser le service Cockpit (port 9090)
firewall-cmd --add-service=cockpit --permanent
firewall-cmd --reload

# Démarrer le service pour Cockpit
systemctl enable --now cockpit.socket
```

- **Il convient finalement d'activer tuned afin que le programme tweak votre système afin d'obtenir les meilleurs performances possibles** :

```
# Démarrer le service tuned
systemctl enable --now tuned

# Choisir le profil adapté à la virtualisation
tuned-adm profile virtual-host
```

- **Une fois l'ensemble de ces opérations réalisées, il est possible d'utiliser Cockpit ou Virt-Manager pour piloter votre hôte de virtualisation** :

Nd0AqRsB5bYa2Tsk-image-1562348718206.png

CCfIdObsWiqFo7AX-image-1562348831179.png

Création d'un bridge pour QEMU/KVM avec NetworkManager et Libvirt

Afin de pouvoir bénéficier d'une connectivité sur un réseau d'entreprise (avec DHCP, DNS...etc), il est nécessaire que les machines virtuelles puissent "parler" sur le réseau. Par défaut, Libvirt crée un réseau de type NAT par défaut, ce qui ne convient pas l'usage décrit précédemment.

Si vous disposez d'un système GNU/Linux disposant de NetworkManager (ex: Fedora Server), il est tout à fait possible de créer un bridge qui va permettre aux machines virtuelles de communiquer sur le réseau de l'entreprise ou votre réseau home. Un bridge Linux est simplement une implémentation de switching Layer 2 au niveau du noyau Linux, il va faire le même travail qu'un vSwitch sur VMware ou un Hyper-V Switch.

Les noms des interfaces sont à adapter en fonction de votre configuration !

- **Création du bridge avec nmcli** :

```
sudo nmcli con add ifname virbr1 type bridge con-name virbr1
```

- **Ajout du bridge à l'interface physique** :

```
sudo nmcli con add type bridge-slave ifname enp3s0 master virbr1
```

- **Éteignez votre interface physique** :

```
sudo nmcli con down "enp3s0"
```

- **Démarrez votre nouveau bridge** :

```
sudo nmcli con up virbr1
```

Une fois ces étapes accomplies, vous devriez voir votre bridge fonctionnel (***ne pas hésiter à redémarrer le service NetworkManager***) :

UNzswOXCL6G0sATY-image-1563188309178.png

Il est également intéressant de déclarer ce bridge à Libvirt pour pouvoir l'utiliser plus facilement avec ce dernier :

- **Créer un fichier "virbr1.xml"** : `vim virbr1.xml`
- **Ajouter le bout de code XML suivant** :

```
<network>
  <name>virbr1</name>
  <forward mode="bridge"/>
  <bridge name="virbr1" />
</network>
```

- **Définissez le réseau à Libvirt** : `sudo virsh net-define virbr1.xml`
- **Démarrez le réseau** : `sudo virsh net-start virbr1`
- **Activer l'autostart du réseau au boot** : `sudo virsh net-autostart virbr1`

Pour éditer des configurations réseaux (IP statique notamment), cela se fait au niveau du bridge ! Pensez à bien flush la configuration de votre interface physique.

That's it !