Virtualisation système et enseignement

Philippe Latu
philippe.latu(at)inetdoc.net

https://www.inetdoc.net

Résumé

Ce guide est à la fois un retour d'expérience et une présentation de la virtualisation de systèmes d'interconnexion réseau dans les enseignements pratiques. L'objectif est de mettre l'accent sur l'identification des fonctions réseau usuelles dans la virtualisation. Dans ce but, on présente la configuration des outils directement depuis l'interface en ligne de commande sans faire appel à une solution de virtualisation intégrée.

Table des matières

1. Copyright et Licence

Méta-information

Cet article est écrit avec DocBook XML sur un système Debian GNU/Linux. Il est disponible en version imprimable aux format PDF : `vm.pdf`.

Toutes les commandes utilisées dans ce document ne sont pas spécifiques à une version particulière des systèmes UNIX ou GNU/Linux. C'est la distribution Debian GNU/Linux qui est utilisée pour les tests présentés.

Conventions typographiques

Tous les exemples d'exécution des commandes sont précédés d'une invite utilisateur ou prompt spécifique au niveau des droits utilisateurs nécessaires sur le système.

- Toute commande précédée de l'invite $ ne nécessite aucun privilège particulier et peut être utilisée au niveau utilisateur simple.

- Toute commande précédée de l'invite # nécessite les privilèges du super utilisateur.

2. Introduction

Depuis quelques années, j'ai pris l'habitude d'utiliser une instance virtuelle de système d'exploitation pour suivre la même démarche que les étudiants lors des séances de travaux pratiques. La première motivation était d'éviter de massacrer la configuration de mon portable à chaque nouvelle séance. Puis, il est apparu que les synthèses intermédiaires faites en reprenant la démarche «qui aurait dû être suivie» par les étudiants permettait de resynchroniser le groupe et d'obtenir des résultats plus homogènes en fin de séance.

Sans démarche de ce genre, une séance de travaux pratiques sur les systèmes informatiques et| ou les réseaux glisse très rapidement vers un bazar incommensurable. En effet, après avoir lu la première question «en diagonale», l'étudiant(e) lambda se jette sur Google™ et se perd dans les méandres de forums rédigés en mode SMS et aux contenus pour le moins contradictoires et douteux. Bref, on en revient toujours à la problématique classique de l'enseignant moyen : comment faire pour que les étudiants adoptent et suivent des méthodes et démarches réfléchies ? Cette lutte contre l'obscurantisme ambiant est particulièrement ardue. Dans l'esprit de beaucoup d'étudiants, l'informatique est une compilation de «recettes de cuisine» sans relations ni cohérence. Dès lors qu'il s'agit d'approfondir ses connaissances et la maîtrise d'un système, les limites de l'apprentissage par «compilation incohérente» apparaissent très vite et les étudiants les moins motivés décrochent très rapidement.

Bien sûr, ce guide sur l'utilisation de la virtualisation ne peut absolument pas prétendre apporter une solution à un problème aussi complexe et difficile. L'objectif, très modeste, est d'apporter un moyen supplémentaire d'illustration de la marche à suivre pour traiter un problème en replaçant bien le contexte.

Il est aujourd'hui très difficile de «détacher» l'esprit de l'auditoire des commandes utilisées à la console pour faire ressortir le processus traité par ces mêmes commandes. Qu'il s'agisse d'enseignements réseau utilisant des systèmes tels qu'IOS ou JunOS ou d'enseignements systèmes utilisant le Shell, les difficultés sont identiques.

- À quel niveau se situe le problème à traiter ? Niveau de la modélisation réseau, couche du système d'exploitation, espace mémoire, etc.

- Quels sont les outils systèmes utilisables à ce niveau ? Configuration d'un protocole, gestionnaire de paquets, analyse de l'occupation mémoire, etc.

- Quelle est la démarche usuelle de diagnostic ? Table de routage, compteurs de liste de contrôle d'accès, liste des processus actifs, journalisation système, etc.

Pour aller un peu plus loin dans l'illustration des méthodes, je souhaite dépasser le stade de la démonstration sur le portable de l'enseignant et fournir aux étudiants des images d'instances de systèmes en fin de séance. Il ne s'agit pas de fournir une solution corrigée ou un produit fini mais plutôt une photo d'un état intermédiaire dans la séance de travaux pratiques. Il devient ainsi possible de proposer aux étudiants de reprendre, non pas la totalité des questions de la séance, mais la partie qui a posé le plus de difficultés.

Voici donc, dans l'état actuel de la réflexion, les choix techniques que j'ai fait pour répondre aux objectifs.

3. Choix d'une solution de virtualisation

Virtualisation et noyau Linux

À l'heure actuelle, il existe deux grandes approches de la virtualisation.

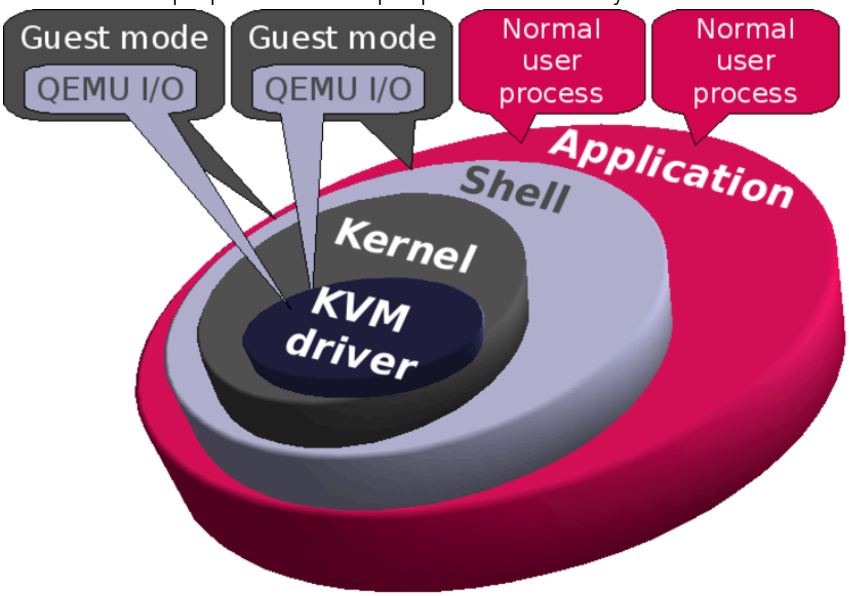
Une première méthode, appelée paravirtualisation, suppose que le noyau du système d'exploitation invité soit modifié pour être exécuté sous forme virtualisée. L'objectif de cette technique est d'offrir un accès quasi identique aux ressources matérielles (mémoire et entrées/sorties) entre système hôte et système invité. Si cet objectif est atteint, les performances du système virtualisé sont vraiment très proches de celles du matériel sur lequel il est exécuté. Le principal obstacle au développement ce cette technique réside justement dans la modification du noyau du système invité. Si le système à virtualiser ne dispose pas de fonctions dédiées à la paravirtualisation dans son noyau, cette technique est inutilisable et la seule solution de virtualisation consiste à utiliser une émulation complète du matériel.

La seconde méthode, appelée virtualisation complète, permet d'exécuter le système d'exploitation invité de manière native sans modification. Avec cette technique, la virtualisation n'a aucun impact sur l'exécution du noyau du système virtualisé. En revanche, la virtualisation complète sacrifie les performances au prix de la compatibilité. En effet, il est plus difficile d'obtenir de bonnes performances lorsque le système invité ne participe pas au processus de virtualisation et doit traverser une ou plusieurs couches d'émulation avant d'accéder aux ressources matérielles.

Côté matériel, les évolutions des fonctions de virtualisation directement intégrées dans les processeurs ont tendance à diminuer les écarts de performances entre les systèmes utilisant la paravirtualisation et les systèmes hôtes. Qu'il s'agisse d'Intel™ (VT) ou d'AMD™ (AMD-V) les processeurs récents disposent de ces fonctions dédiées à la virtualisation.

Avec le noyau Linux, l'objectif de la solution Kernel Based Virtual Machine ou KVM est d'ajouter des capacités de virtualisation à un noyau standard.

Avec le modèle KVM, chaque machine virtuelle est un processus standard du noyau Linux géré par l'ordonnanceur (scheduler). Un processus normal de système GNU/Linux peut être exécuté selon deux modes : noyau (kernelspace) ou utilisateur (userspace). Le modèle KVM ajoute un troisième mode : le mode invité qui possède ses propres modes noyau et utilisateur.



KVM dans le noyau Linux - vue complète

Le modèle de virtualisation KVM comprend deux composants.

- Un pilote de périphérique pour gérer le matériel virtualisé. Les fonctions de ce pilote sont disponibles via le fichier spécial d'interface de type caractère `/dev/kvm`.
- Un logiciel utilisateur pour émuler le matériel d'un PC. Cette partie utilisateur (userspace) est fournie par les paquets de la famille QEMU qui sont présentés à la la section intitulée « Outils nécessaires ».

Côté paravirtualisation, ce sont les fonctions virtio du noyau Linux qui permettent d'utiliser des canaux de communication dédiés entre instances de machines virtuelles et système hôte. Depuis la sortie de la version stable baptisée Lenny de la distribution Debian GNU/Linux, ces fonctions sont

disponibles dans les paquets de noyau. Aucune manipulation n'est donc nécessaire pour bénéficier de la paravirtualisation avec une instance de système Debian GNU/Linux.

Contexte pratique

KVM, Kernel-based Virtual Machine

> Kernel Based Virtual Machine est devenue rapidement la solution de virtualisation de référence pour Linux. Elle est basée sur les architectures Intel™ baptisées Intel VT™ ou les architectures AMD™ baptisées AMD-V™. Ces deux familles de processeurs possèdent les extensions matérielles de virtualisation. Un module du noyau Linux fournit le cœur de virtualisation et un module spécifique dépendant du processeur (kvm-intel.ko ou kvm-amd.ko) active les fonctions matérielles.

QEMU

> QEMU open source machine emulator est un émulateur machine générique. Il peut exécuter des instances de systèmes d'exploitation et des programmes faits pour une architecture processeur particulière (carte mère ARM par exemple) sur une machine hôte d'architecture différente (votre propre PC par exemple). En utilisant la traduction dynamique, QEMU donne de bonnes performances.
>
> QEMU est la solution de virtualisation à utiliser si le processeur de système hôte ne possède pas d'extension matérielle spécifique à la virtualisation.

Les développements de ces deux outils distincts progressent en parallèle et ils utilisent le même jeu d'options de configuration.

La présentation de ces 2 outils complémentaires introduit la distinction entre les processeurs possédant les extensions de virtualisation ou non. Pour identifier les caractéristiques des processeurs avec un système hôte GNU/Linux, on utilise les informations fournies dans l'arborescence `/proc`. Si le seul système d'exploitation installé sur la machine hôte est de type Windows, il est toujours possible de lancer la machine en utilisant un live CD de type KNOPPIX pour effectuer la même opération.

Suivant le résultat de la commande suivante, on sait si le processeur peut utiliser la solution KVM ou non.

```
$ egrep '(vmx|svm)' /proc/cpuinfo
```

Si la commande ne donne aucun résultat, le processeur ne possède pas d'extension de virtualisation et seule la solution QEMU est utilisable.

Si le résultat donne une ou plusieurs lignes (suivant le nombre de cœurs du processeur) débutant par `flags`, la solution KVM est utilisable avec les outils QEMU.

4. KVM et VIRTIO

Comme on l'a vu dans la Section 3, « Choix d'une solution de virtualisation », ces bibliothèques permettent à une instance de machine virtualisée d'utiliser des canaux de communications particuliers vers le matériel du système hôte. Parmi ces canaux, on trouve les accès : mémoire, disque, horloge temps réel et réseau.

Identification des modules disponibles

Les modules autorisant l'utilisation de ces bibliothèques dans l'espace utilisateur sont disponibles dans les paquets de noyau Linux. Voici quelques exemples d'identification des fonctions virtio sur un système virtualisé.

On recherche la version du noyau en cours d'exécution.

```
$ uname -a
Linux vm0 4.14.0-3-amd64 #1 SMP Debian 4.14.13-1 (2018-01-14) x86_64 GNU/Linux

$ aptitude search ~ilinux-image
i A linux-image-4.14.0-3-amd64  - Linux 4.14 pour les ordinateurs 64 bits
i   linux-image-amd64          - Linux pour les ordinateurs 64 bits (métapaquet)
```

On liste les modules relatifs aux fonctions `Virtio` à partir de la version du noyau identifiée.

```
$ find /lib/modules/$(uname -r)/ -type f -name '*virtio*'
/lib/modules/4.14.0-3-amd64/kernel/net/9p/9pnet_virtio.ko
/lib/modules/4.14.0-3-amd64/kernel/net/vmw_vsock/vmw_vsock_virtio_transport.ko
/lib/modules/4.14.0-3-amd64/kernel/net/vmw_vsock/vmw_vsock_virtio_transport_common.ko
/lib/modules/4.14.0-3-amd64/kernel/drivers/net/virtio_net.ko
/lib/modules/4.14.0-3-amd64/kernel/drivers/virtio/virtio_pci.ko
/lib/modules/4.14.0-3-amd64/kernel/drivers/virtio/virtio_balloon.ko
/lib/modules/4.14.0-3-amd64/kernel/drivers/virtio/virtio_input.ko
/lib/modules/4.14.0-3-amd64/kernel/drivers/virtio/virtio_ring.ko
/lib/modules/4.14.0-3-amd64/kernel/drivers/virtio/virtio.ko
/lib/modules/4.14.0-3-amd64/kernel/drivers/gpu/drm/virtio/virtio-gpu.ko
/lib/modules/4.14.0-3-amd64/kernel/drivers/crypto/virtio/virtio_crypto.ko
/lib/modules/4.14.0-3-amd64/kernel/drivers/block/virtio_blk.ko
/lib/modules/4.14.0-3-amd64/kernel/drivers/char/hw_random/virtio-rng.ko
/lib/modules/4.14.0-3-amd64/kernel/drivers/char/virtio_console.ko
/lib/modules/4.14.0-3-amd64/kernel/drivers/scsi/virtio_scsi.ko
```

Utilisation des bibliothèques VIRTIO

Comme on l'a dit précédemment, ces bibliothèques constituent des canaux de communication avec le matériel du système hôte. Elles ont donc un effet sur la représentation du matériel.

À titre d'exemple, voici la liste des périphériques «visibles» sur le bus PCI d'une instance de système virtuel.

```
$ lspci
00:00.0 Host bridge: Intel Corporation 82G33/G31/P35/P31 Express DRAM Controller
00:01.0 VGA compatible controller: Red Hat, Inc. QXL paravirtual graphic card (rev 04)
00:02.0 Audio device: Intel Corporation 82801FB/FBM/FR/FW/FRW (ICH6 Family) High Definition Audio Controller (rev 0
00:03.0 Unclassified device [00ff]: Red Hat, Inc Virtio memory balloon
00:04.0 SCSI storage controller: Red Hat, Inc Virtio block device
00:05.0 Communication controller: Red Hat, Inc Virtio console
00:06.0 Ethernet controller: Red Hat, Inc Virtio network device
00:07.0 System peripheral: Intel Corporation 6300ESB Watchdog Timer
00:1d.0 USB controller: Intel Corporation 82801I (ICH9 Family) USB UHCI Controller #1 (rev 03)
00:1d.1 USB controller: Intel Corporation 82801I (ICH9 Family) USB UHCI Controller #2 (rev 03)
00:1d.2 USB controller: Intel Corporation 82801I (ICH9 Family) USB UHCI Controller #3 (rev 03)
00:1d.7 USB controller: Intel Corporation 82801I (ICH9 Family) USB2 EHCI Controller #1 (rev 03)
00:1f.0 ISA bridge: Intel Corporation 82801IB (ICH9) LPC Interface Controller (rev 02)
00:1f.2 SATA controller: Intel Corporation 82801IR/IO/IH (ICH9R/DO/DH) 6 port SATA Controller [AHCI mode] (rev 02)
00:1f.3 SMBus: Intel Corporation 82801I (ICH9 Family) SMBus Controller (rev 02)
```

Dans la liste ci-dessus, on reconnaît 4 périphériques référencés `Virtio` dont le gestionnaire mémoire et le contrôleur réseau.

Sur le même système, on peut lister les modules chargés en mémoire relatifs aux bibliothèques.

```
$ lsmod | grep virtio
virtio_console         32768  0
virtio_balloon         20480  0
virtio_net             49152  0
virtio_blk             20480  3
virtio_pci             28672  0
virtio_ring            28672  5 virtio_blk,virtio_net,virtio_balloon,virtio_console,virtio_pci
virtio                 16384  5 virtio_blk,virtio_net,virtio_balloon,virtio_console,virtio_pci
```

Cette liste correspond aux fonctionnalités utilisées par le système virtuel relativement au catalogue donné dans la section précédente.

5. Création d'une machine virtuelle

Note

Toutes les opérations présentées dans cette section utilisent le répertoire utilisateur `~/vm/` dans lequel sont stockées les images de systèmes virtuels.

Outils nécessaires

Sachant que les modules de virtualisation appartenant au noyau Linux sont chargés en mémoire, il faut maintenant connaître la liste des outils nécessaires à la création et au lancement des instances de machines virtuelles. Les paquets associés à la gestion des images de machines virtuelles sont les suivants :

```
$ aptitude search ~iqemu
i   ipxe-qemu                 - PXE boot firmware - ROM images for qemu
i   qemu-system-common        - QEMU full system emulation binaries (common files)
i   qemu-system-x86           - QEMU full system emulation binaries (x86)
i   qemu-utils                - QEMU utilities
```

Le paquet qemu-system-x86 correspond au processus d'émulation d'architecture x86 avec le support des extensions matérielles : Intel VT™ et AMD SVM™. Il fournit aussi les éléments de description des périphériques de carte mère pour cette architecture : contrôleur PCI, carte vidéo, carte réseau, carte son, etc.

Installation d'une machine virtuelle

On commence par créer un fichier image de disque virtuel qui servira de support au système de fichiers de la nouvelle instance de système d'exploitation.

```
:~/vm$ qemu-img create -f raw vm0-debian-stable-amd64.raw 80G
Formatting 'vm0-debian-stable-amd64.raw', fmt=raw size=85899345920

:~/vm$ du -hs vm0-debian-stable-amd64.raw
0       vm0-debian-stable-amd64.raw
```

Le format d'image créé à l'aide de l'instruction ci-dessus est baptisé `raw`. Il s'agit d'un format brut, utilisé par défaut, sur lequel seuls les secteurs écrits entraînent une réservation d'espace. Ce choix de format peut être modifié par la suite puisqu'il est toujours possible de convertir une image d'un format à un autre après coup.

On utilise cette image disque pour lancer le processus d'installation de la machine virtuelle.

```
:~/vm$ ./scripts/iso-install-startup.sh \
vm0-debian-stable-amd64.raw \
~/iso.images/debian-8.3.0-amd64-netinst.iso \
512 \
0

~> Machine virtuelle : vm0-debian-stable-amd64.raw
~> Port SPICE        : 5900
~> Mémoire RAM       : 512
~> Adresse MAC       : ba:ad:ca:fe:00:00
```

Pour l'installation du système d'exploitation de la machine virtuelle, on utilise le script iso-install-startup.sh donné en annexe Section A.4, « Installation d'un système sur machine virtuelle ». Ce script impose au moins quatre paramètres.

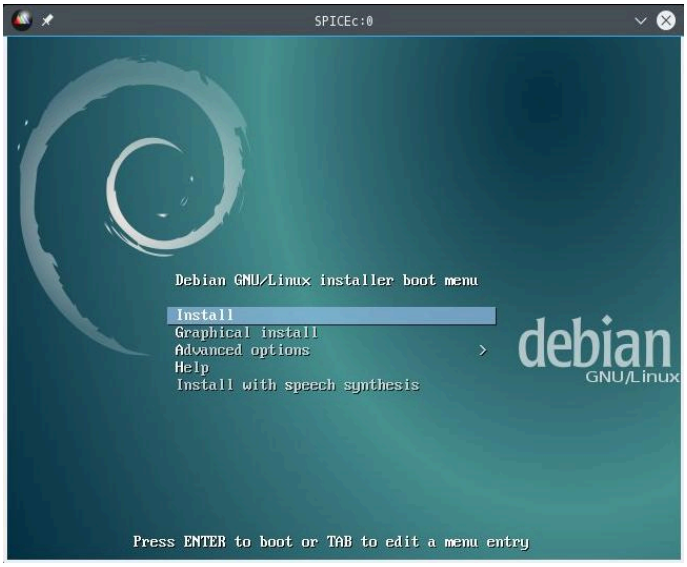
1. Le fichier image de la machine virtuelle : `vm0-debian-stable-amd64.raw` dans l'exemple ci-dessus.

2. Le fichier au format `iso` correspondant au média d'installation. Dans l'exemple, il s'agit de l'image `netinst` de la version Jessie du système Debian GNU/Linux.

3. La quantité de RAM allouée à la machine virtuelle : 512Mo dans l'exemple.

4. Le numéro du port d'accès à l'écran de la machine virtuelle via le protocole SPICE. Ici la valeur `0` conduit à l'utilisation du port numéro `5900`.

Après l'initialisation de la machine virtuelle, le processus d'installation classique démarre.



Écran d'installation Debian GNU/Linux - vue complète

Utilisation du protocole SPICE

Le protocole SPICE a pour but de fournir un accès distant interactif à l'écran d'un système virtualisé. Il s'agit d'un protocole client/serveur dont la partie serveur est située sur la machine virtuelle.

Dans le contexte de ce guide, les paramètres nécessaires à l'utilisation de ce protocole sont présents dans les scripts proposés en Annexe A, Scripts spécifiques. Un choix de paramétrage particulier a été fait. L'accès est ouvert sur l'adresse localhost du système hôte dans le but de faciliter l'utilisation de l'écran distant avec IPv4 ou IPv6. Cet accès ne devient «vraiment distant» que si un tunnel SSH est ouvert entre le système hôte et le système sur lequel on souhaite afficher l'écran de la machine virtuelle.

Ainsi, le script iso-install-startup.sh utilisé plus haut ouvre un accès sur le port `tcp/5900` en écoute sur l'interface de boucle locale du système hôte. On peut établir un tunnel SSH puis exécuter le client SPICE à l'aide des commandes suivantes.

```
:~$ ssh -f -N -L 5900:localhost:5900 <user>@<hostname>

:~$ spicec -h localhost -p 5900
```

Le programme spicec appartient au paquet spice-client.

C'est à l'aide de ces paramètres que le protocole SPICE a été utilisé pour réaliser la copie d'écran ci-dessus.

6. Optimisation système

Cette section est une liste de quelques trucs et astuces permettant d'optimiser l'usage d'une instance virtuelle de système d'exploitation.

Gestion de paquets

Pour optimiser la gestion de paquets avec APT sur une machine virtuelle Debian GNU/Linux on essaie de limiter au maximum l'occupation disque de façon à donner un maximum d'espace dans l'arborescence `/var`.

`# aptitude clean`

Il est inutile de conserver les fichiers `.deb` correspondant aux paquets installés sur le système virtuel. Cette commande sert justement à nettoyer l'arborescence `/var/cache/apt`.

`# aptitude purge $(deborphan)`

La commande deborphan appartenant au paquet du même nom recherche les paquets orphelins installé sur le système. Les paquets trouvés cette commande ne sont pas nécessaires au

fonctionnement des services installés ; on peut donc les supprimer sans problème pour gagner de la place disque.

`# localepurge`

La commande localepurge appartenant au paquet du même nom. Elle sert à effacer tous les fichiers de «localisation» (langues étrangères) inutiles sur le disque. Elle est appelée automatiquement à chaque opération de gestion de paquets avec `apt-get` ou `aptitude`.

La fin du fichier de configuration `/etc/locale.nopurge` répertorie les paramètres des fichiers de localisation à conserver.

```
# tail /etc/locale.nopurge

#VERBOSE

############################################################
# Following locales won't be deleted from this system
# after package installations done with apt-get(8):

en
fr
fr_FR.UTF-8
```

Transparent proxy, Service Mandataire

L'utilisation du gestionnaire de paquets peut poser problème lorsque le système hôte se trouve derrière un service mandataire ou proxy transparent.

Dans ce cas, il faut compléter la configuration du gestionnaire de paquets sur chaque instance de système virtuel en ajoutant un fichier dans le répertoire `/etc/apt/apt.conf.d/`.

```
# cat /etc/apt/apt.conf.d/10proxy
Acquire::http::No-Cache "true";
Acquire::http::Max-Age "0";
```

Duplication du jeu de paquets entre systèmes, aptitude-create-state-bundle, aptitude-run-state-bundle

Pour dupliquer le jeu de paquets installés entre instances système, il est possible de créer un fichier image de l'état d'une installation. Cette technique permet de transférer la liste des paquets installés entre architectures différentes.

Côté système source, celui qui détient le jeu de référence, on utilise la commande aptitude-create-state-bundle et copie le fichier contenant la liste des paquets installés sur le système hôte.

```
# aptitude-create-state-bundle selections.bz2
# chown etu.etu selections.bz2
# exit
etu@vm:~$ logout
Connection to fe80::b8ad:caff:fefe:5%vlan5 closed
:~/vm$ $ scp etu@[fe80::b8ad:caff:fefe:5%vlan5]:~/selections.bz2 .
Warning: Permanently added 'fe80::b8ad:caff:fefe:5%vlan5' (ECDSA) to the list of known hosts.
etu@fe80::b8ad:caff:fefe:5%vlan5's password:
selections.bz2                100%   30MB  30.4MB/s   00:01
```

Côté système cible, celui sur lequel on doit appliquer le jeu de référence pour compléter sa liste de paquets installés, on utilise la commande aptitude-run-state-bundle.

```
:~/vm$ scp selections.bz2 etu@[fe80::b8ad:caff:fefe:0%vlan214]:~
Warning: Permanently added 'fe80::b8ad:caff:fefe:0%vlan214' (ECDSA) to the list of known hosts.
etu@fe80::b8ad:caff:fefe:0%vlan214's password:
selections.bz2                100%   30MB  30.4MB/s   00:00
:~/vm$ ssh etu@fe80::b8ad:caff:fefe:0%vlan214
Warning: Permanently added 'fe80::b8ad:caff:fefe:0%vlan214' (ECDSA) to the list of known hosts.
etu@fe80::b8ad:caff:fefe:0%vlan214's password:

etu@vm:~$ su
Mot de passe :
vm:/home/etu# aptitude-run-state-bundle selections.bz2
```

Paramétrage de la console

Que l'on accède à une instance de système virtualisé via une connexion SSH ou via un écran distant avec le protocole SPICE, il est important de disposer d'un jeu de caractères et d'un affichage cohérent.

Terminal d'exécution, gpm

Le paquet gpm permet d'utiliser la souris en mode console pour effectuer les opérations du type copier/coller.

`keyboard-configuration`, `console-setup`

Le paquet keyboard-configuration permet de gérer les codes de touches clavier et les jeux de caractères. La reconfiguration du paquet avec la commande `# dpkg-reconfigure keyboard-configuration` permet de passer en revue et de modifier les options relatives aux dispositions de claviers.

Le paquet console-setup permet de paramétrer la configuration du clavier et de la police pour la console du système.

`spice-vdagent`

Dans le cas où l'on accède à un écran graphique de système virtualisé via le protocole SPICE, il est possible de «partager» les actions à la souris entre l'interface graphique du système local et du système virtuel. Cet outil permet aussi le redimensionnement dynamique de l'écran du système virtuel en fonction de la résolution de l'écran du système local.

Configuration graphique

La gestion de l'interface graphique d'une instance de système virtualisé est en constante sur les hyperviseurs disponibles avec le noyau Linux. Dans ce document, on se réfère au protocole SPICE et au contrôleur graphique virtuel `QXL`.

Sur un système non virtualisé le graphisme utilise les ressources d'un composant spécifique baptisé Graphics Processing Unit ou GPU. La virtualisation implique l'émulation d'un composant GPU. Cette émulation doit être en mesure de traiter tous les appels aux bibliothèques graphiques effectués par les différents gestionnaire d'interface utilisateur. La tâche est donc particulièrement ardue.

Les scripts utilisés pour la rédaction de ce guide désignent le contrôleur QXL comme GPU. Voici quelques éléments d'identification.

```
$ lspci | grep VGA
00:01.0 VGA compatible controller: Red Hat, Inc. Device 0100 (rev 04)
```

Malheureusement, la version stable Wheezy ne dispose pas du paquet de pilotage de GPU émulé. Le suuport des fonctions graphiques est donc moins performant. En revanche, la version testing contient bien le paquet xserver-xorg-video-qxl.

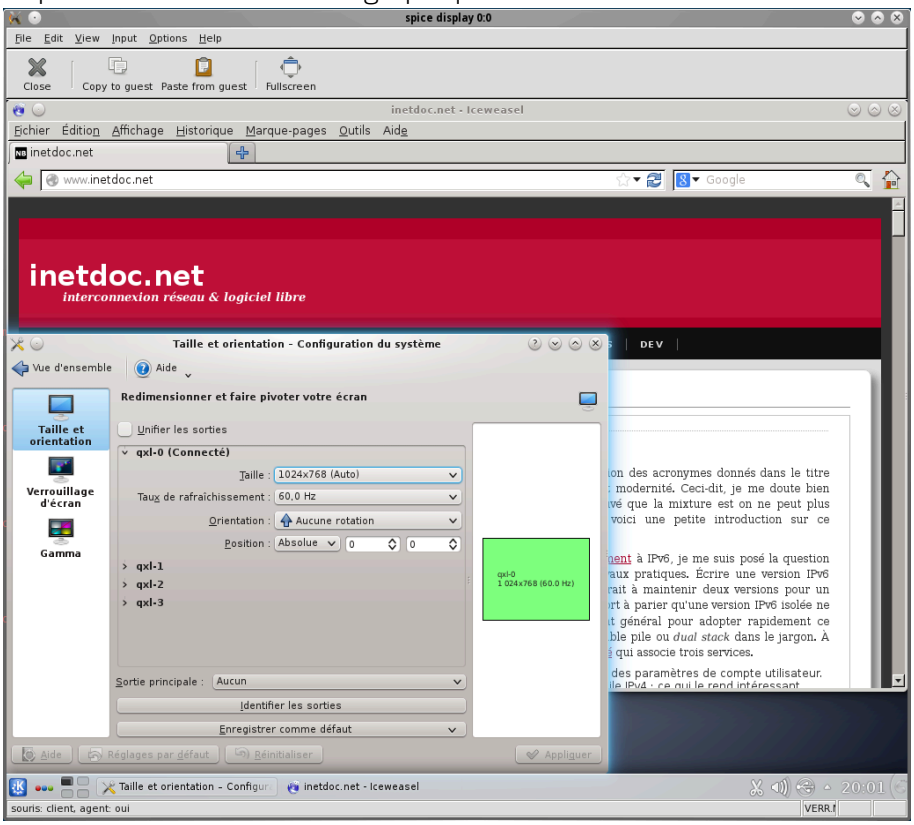
- Extrait du journal d'initialisation du serveur graphique sur la distribution stable Wheezy.

```
# grep -i qxl /var/log/Xorg.0.log
[    10.785] (==) Matched qxl as autoconfigured driver 0
[    10.785] (II) LoadModule: "qxl"
[    10.848] (WW) Warning, couldn't open module qxl
[    10.848] (II) UnloadModule: "qxl"
[    10.848] (II) Unloading qxl
[    10.848] (EE) Failed to load module "qxl" (module does not exist, 0)
```

- Extrait du journal d'initialisation du serveur graphique sur la distribution testing Jessie.

```
# grep -i qxl /var/log/Xorg.0.log
[    13.206] (==) Matched qxl as autoconfigured driver 0
[    13.206] (II) LoadModule: "qxl"
[    13.230] (II) Loading /usr/lib/xorg/modules/drivers/qxl_drv.so
[    13.240] (II) Module qxl: vendor="X.Org Foundation"
[    13.252] (II) qxl: Driver for QXL virtual graphics: QXL 1
[    13.302] (II) qxl(0): Creating default Display subsection in Screen section
[    13.302] (==) qxl(0): Depth 24, (--) framebuffer bpp 32
[    13.302] (==) qxl(0): RGB weight 888
[    13.302] (==) qxl(0): Default visual is TrueColor
[    13.302] (==) qxl(0): Using gamma correction (1.0, 1.0, 1.0)
[    13.302] (II) qxl(0): Offscreen Surfaces: Enabled
[    13.302] (II) qxl(0): Image Cache: Enabled
[    13.302] (II) qxl(0): Fallback Cache: Enabled
[    13.303] (II) qxl(0): framebuffer at 0x7f1d30471000 (16384 KB)
[    13.303] (II) qxl(0): command ram at 0x7f1d31471000 (32760 KB)
[    13.303] (II) qxl(0): vram at 0x7f1d2c471000 (65536 KB)
[    13.303] (II) qxl(0): rom at 0x7f1d3852a000
[    13.303] (II) qxl(0): Device version 0.0
[    13.303] (II) qxl(0): Compression level 0, log level 0
[    13.303] (II) qxl(0): 12286 io pages at 0x7f1d30471000
[    13.303] (II) qxl(0): RAM header offset: 0x3ffe000
[    13.303] (II) qxl(0): Correct RAM signature 41525851
[    13.303] (II) qxl(0): 49144 KB of video RAM
[    13.303] (II) qxl(0): 1024 surfaces
```

Copie d'écran de l'interface graphique sur une instance de machine virtuelle Debian/testing.



Pilote graphique QXL - copie d'écran

Système de fichiers ext4

Il est possible d'optimiser les accès au système de fichiers ext4 sur un système virtualisé. Dans la mesure où l'on n'accède pas à un véritable dispositif de stockage, certaines options de montage peuvent être modifiées. Voici un extrait de fichier `/etc/fstab` qui donne la liste des options de montage de la racine du système de fichiers.

```
etu@vm0:~$ grep ext4 /etc/fstab
/dev/mapper/vm0-root /   ext4    relatime,data=writeback,commit=6000,barrier=0,errors=remount-ro 0 1
```

`relatime`

Cette option assure le maintien des méta-données du système de fichiers relatives aux accès. L'enregistrement `atime` d'un fichier est écrit seulement si le le fichier a été modifié depuis la dernière

mise à jour de cet enregistrement (`mtime`) ou si aucun accès au fichier n'a eu lieu pendant un certain temps.

`data=writeback`

Le séquencement des données n'est pas préservé. Les données peuvent être écrites dans le système de fichiers principal après que les méta-données aient été enregistrées dans le journal.

Pour activer cette option pour un système de fichiers particulier, la syntaxe est :

```
# tune2fs -o journal_data_writeback /dev/mapper/vm0-root
tune2fs 1.42.9 (4-Feb-2014)

# tune2fs -l /dev/mapper/vm0-root | grep '^Default mount options:'
Default mount options:    journal_data_writeback
```

`commit=6000`

Cette option contrôle de la synchronisation des méta-données du système de fichiers toutes les 6000 secondes.

`barrier=0`

Cette option contrôle l'ordonnancement dans le cache du système de fichiers. Sur une machine virtuelle, il n'est pas nécessaire d'exercer un contrôle à ce niveau.

Accès SSH

Pour administrer des systèmes à distance, le protocole SSH est le mode de connexion universel. Dans le contexte particulier de l'administration des instances de systèmes virtuels, une même image système avec une même clé d'hôte RSA est dupliquée autant de fois que nécessaire. On se trouve rapidement confronté aux traditionnels messages d'alerte sur l'usurpation de cette clé d'hôte lorsque l'on accès aux instances virtuelles depuis le système hôte.

On peut configurer le client SSH du système hôte de façon à ne pas vérifier «l'identité», c'est à dire la clé d'hôte RSA, des instances de systèmes virtualisées. La documentation complète sur la configuration du client SSH est fournie dans les pages de manuels : `$ man ssh_config`.

Voici un extrait de fichier `~/.ssh/config` qui désactive les contrôles pour tous les hôtes dont les adresses IP correspondent au masque `192.0.2.*`.

```
Host 192.0.2.*
  CheckHostIP no
  StrictHostKeyChecking no
  UserKnownHostsFile=/dev/null
```

7. Extension de la capacité de stockage avec LVM

L'énorme avantage de l'utilisation du gestionnaire de volume logique (Logical Volume Manager ou LVM) avec les images disques de machine virtuelle, c'est que l'on peut manipuler la capacité de stockage après installation.

Du point de vue pédagogique, c'est aussi l'occasion de se familiariser avec les fonctionnalités offertes par les outils LVM. Ces outils sont devenus indispensables dans la gestion de la capacité de stockage.

Voici un exemple d'extension de la capacité de stockage d'une image de machine virtuelle.

Extension du fichier image

On commence par identifier les caractéristiques du fichier image vu du système hôte. Le fichier au format `raw` est un sparse file. Si on a réservé un volume total, seule l'occupation effective est prise en compte pour le calcul de l'occupation disque.

La commande ls fait apparaître l'occupation effective ainsi que l'espace total réservé lors de la création du fichier image.

```
$ ls -gGh
total 3,4G
-rw-r--r-- 1 32G janv. 13 18:58 vm0-debian-testing-amd64-base.raw
```

Une fois l'instance de système virtuel lancée, on obtient l'occupation disque suivante.

```
etu@vm0:~$ df -h
Sys. de fichiers     Taille Utilisé Dispo Uti% Monté sur
/dev/mapper/vm0-root    30G    937M   28G   4% /
udev                    10M       0   10M   0% /dev
tmpfs                  202M    204K  201M   1% /run
tmpfs                  5,0M       0  5,0M   0% /run/lock
tmpfs                  403M       0  403M   0% /run/shm
/dev/vda1              228M     23M  190M  11% /boot
```

Toujours à partir de la même instance la table des partitions est la suivante.

```
# parted /dev/vda print
Model: Virtio Block Device (virtblk)
Disk /dev/vda: 34,4GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End     Size    Type      File system  Flags
 1      1049kB  256MB   255MB   primary   ext2         boot
 2      257MB   34,4GB  34,1GB  extended
 5      257MB   34,4GB  34,1GB  logical               lvm
```

L'opération d'extension, s'effectue directement sur le fichier image au format «brut» non compressé `raw`.

```
$ qemu-img resize vm0-debian-testing-amd64-base.raw +20G
Image resized.
```

Extension de l'espace de stockage du système virtuel

En utilisant l'image système étendue générée ci-avant, on fait apparaître l'espace disponible en affichant l'état des partitions.

```
# parted /dev/vda print
Model: Virtio Block Device (virtblk)
Disk /dev/vda: 55,8GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End     Size    Type      File system  Flags
 1      1049kB  256MB   255MB   primary   ext2         boot
 2      257MB   34,4GB  34,1GB  extended
 5      257MB   34,4GB  34,1GB  logical               lvm
```

On remarque que l'espace total a bien été augmenté et qu'aucune partition n'est disponible. Il faut donc créer une nouvelle partition correspondant à l'espace libre.

```
# parted /dev/vda unit % print free
Model: Virtio Block Device (virtblk)
Disk /dev/vda: 100%
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start  End    Size   Type      File system  Flags
        0,00%  0,00%  0,00%            Free Space
 1      0,00%  0,46%  0,46%  primary   ext2         boot
        0,46%  0,46%  0,00%            Free Space
 2      0,46%  61,5%  61,1%  extended
 5      0,46%  61,5%  61,1%  logical               lvm
        61,5%  100%   38,5%            Free Space

# parted /dev/vda mkpart primary 62% 100%
Information: You may need to update /etc/fstab.

# parted /dev/vda set 3 lvm on

# parted /dev/vda print
Model: Virtio Block Device (virtblk)
Disk /dev/vda: 55,8GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End     Size    Type      File system  Flags
 1      1049kB  256MB   255MB   primary   ext2         boot
 2      257MB   34,4GB  34,1GB  extended
 5      257MB   34,4GB  34,1GB  logical               lvm
 3      34,4GB  55,8GB  21,5GB  primary               lvm
```

C'est la nouvelle partition `/dev/vda3` que nous utilisons pour créer un nouveau volume physique LVM. On visualise ensuite les propriétés des deux volumes physiques du système virtuel.

```
# pvcreate /dev/vda3
  Physical volume "/dev/vda3" successfully created

# pvdisplay
  --- Physical volume ---
  PV Name               /dev/vda5
  VG Name               vm0
  PV Size               31,76 GiB / not usable 2,00 MiB
  Allocatable           yes (but full)
  PE Size               4,00 MiB
  Total PE              8130
  Free PE               0
  Allocated PE          8130
  PV UUID               CpaZ5D-vbVS-32w3-QLnk-GVAd-06pB-y2Iw8Y

  "/dev/vda3" is a new physical volume of "20,00 GiB"
  --- NEW Physical volume ---
  PV Name               /dev/vda3
  VG Name
  PV Size               20,00 GiB
  Allocatable           NO
  PE Size               0
  Total PE              0
  Free PE               0
  Allocated PE          0
  PV UUID               jZJ5xq-bZlt-m0ZL-RkUG-Phe2-bbDY-r6cAPI
```

On remarque les différences entre les deux volumes physiques LVM. Le volume physique `/dev/vda3` n'est associé à aucun groupe de volumes et ne dispose pas d'une taille de bloc ou Physical Extent (PE) définie. L'étape suivante consiste à étendre le groupe de volumes logiques avec le nouveau volume physique disponible ; c'est le rôle de la commande vgextend. On visualise ensuite le résultat sur les propriétés du nouveau volume physique.

```
# vgextend vm0 /dev/vda3
  Volume group "vm0" successfully extended

# pvdisplay /dev/vda3
  --- Physical volume ---
  PV Name               /dev/vda3
  VG Name               vm0
  PV Size               20,00 GiB / not usable 0
  Allocatable           yes
  PE Size               4,00 MiB
  Total PE              5120
  Free PE               5120
  Allocated PE          0
  PV UUID               jZJ5xq-bZlt-m0ZL-RkUG-Phe2-bbDY-r6cAPI
```

Pour achever l'opération, on affecte l'espace offert par le nouveau volume physique à un volume logique du système. Dans l'exemple ci-dessous, on se propose de déplacer le contenu de l'arborescence `/var` dans le nouveau volume logique.

```
# lvcreate --name var vm0 /dev/vda3 -l 100%FREE
  Logical volume "var" created

# lvdisplay /dev/vm0/var
  --- Logical volume ---
  LV Path                /dev/vm0/var
  LV Name                var
  VG Name                vm0
  LV UUID                mQTwlc-4zNt-0V7P-SfWM-qh3G-Zulh-y4zYN4
  LV Write Access        read/write
  LV Creation host, time vm0, 2014-01-15 00:20:40 +0100
  LV Status              available
  # open                 0
  LV Size                20,00 GiB
  Current LE             5120
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           253:2
```

Une fois le nouveau volume logique prêt à l'emploi, on passe aux manipulations sur le système de fichiers avec le formatage, la synchronisation de l'arborescence et la définition du point de montage.

```
# mkfs.ext4 /dev/mapper/vm0-var
mke2fs 1.42.9 (28-Dec-2013)
Étiquette de système de fichiers=
Type de système d'exploitation : Linux
Taille de bloc=4096 (log=2)
Taille de fragment=4096 (log=2)
« Stride » = 0 blocs, « Stripe width » = 0 blocs
1310720 i-noeuds, 5242880 blocs
262144 blocs (5.00%) réservés pour le super utilisateur
Premier bloc de données=0
Nombre maximum de blocs du système de fichiers=4294967296
160 groupes de blocs
32768 blocs par groupe, 32768 fragments par groupe
8192 i-noeuds par groupe
Superblocs de secours stockés sur les blocs :
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
        4096000

Allocation des tables de groupe : complété
Écriture des tables d'i-noeuds : complété
Création du journal (32768 blocs) : complété
Écriture des superblocs et de l'information de comptabilité du système de
fichiers : complété

# mount /dev/mapper/vm0-var /mnt

# rsync -ah --numeric-ids /var/ /mnt/

# umount /mnt
```

Enfin, on créé l'entrée pour le nouveau point de montage dans le fichier `/etc/fstab`.

```
# grep var /etc/fstab
/dev/mapper/vm0-var /var        ext4    noatime,commit=600,defaults        0       2
```

En conclusion, on visualise l'espace occupé et disponible sur l'ensemble des volumes logiques du système virtuel après redémarrage.

```
# df -h
Sys. de fichiers     Taille Utilisé Dispo Uti% Monté sur
/dev/mapper/vm0-root    30G    742M   28G   3% /
udev                    10M       0   10M   0% /dev
tmpfs                  202M    212K  201M   1% /run
tmpfs                  5,0M       0  5,0M   0% /run/lock
tmpfs                  403M       0  403M   0% /run/shm
/dev/vda1              228M     23M  190M  11% /boot
/dev/mapper/vm0-var     20G    242M   19G   2% /var
```

Bien sûr, les manipulations réalisées dans cette section ne sont que des exemples. Il est possible de faire beaucoup d'autres manipulations avec le gestionnaire de volume logique LVM. L'extension de capacité n'est qu'une facette des fonctionnalités offertes.

8. Communications réseau en mode utilisateur

Le mode utilisateur correspond à l'utilisation d'une pile de protocoles dans l'espace mémoire utilisateur. L'avantage de cette solution, c'est que la configuration intégrée ne nécessite aucun paramétrage réseau sur le système virtualisé dès lors que l'on opte pour l'autoconfiguration IPv6 ou pour le service DHCP avec IPv4 lors de l'installation. L'inconvénient, c'est qu'aucune communication réseau vers la machine virtuelle n'est possible. Il est cependant possible d'accéder à la «console» d'une machine virtuelle en mode utilisateur depuis le système hôte à l'aide d'une interface réseau factice qui utilise l'option `hostfwd`.

La documentation officielle se trouve à la page Using the user mode network stack

Selon les termes de cette documentation, la machine virtuelle se comporte comme un hôte réseau situé derrière un pare-feu qui bloque les connexions entrantes. La topologie de la connexion et des services se présente sous la forme suivante :

```
QEMU VLAN      <------>  Firewall/DHCP server <-----> Internet
                           |          (10.0.2.2)
                           |
                           ---->  DNS server (10.0.2.3)
                           |
                           ---->  SMB server (10.0.2.4)
```

Ce mode de fonctionnement intégré à la solution de virtualisation reproduit les services traditionnellement offerts par un routeur domestique ADSL : traduction d'adresses IP (NAT), configuration réseau cliente dynamique (DHCP) et résolution des noms de domaines (DNS).

Le serveur virtuel attribue dynamiquement les adresses IP à partir de `10.0.2.15` aux interfaces réseau de la machine virtuelle.

Exemple d'utilisation des communications réseau en mode utilisateur

On lance l'instance de machine virtuelle à l'aide du script standalone-startup.sh donné en annexe Section A.1, « Communications réseau en mode utilsateur ».

```
:~/vm$./scripts/standalone-startup.sh vm0-debian-stable-amd64-base.raw 512 10

~> Machine virtuelle : vm0-debian-stable-amd64-base.raw
~> Port SPICE        : 5910
~> Mémoire RAM       : 512
~> Adresse MAC       : ba:ad:ca:fe:00:0a
```

Les options réseau intéressantes de ce script sont les suivantes.

```
-device virtio-net,netdev=net0,mac="$macaddress" \
-netdev user❶,id=net0,hostfwd=tcp::$((5900 + $tapnum))-:22  ❷
```

❶ Le paramètre `user` applique la configuration réseau intégrée à l'interface nommée `net0`.

❷ L'option `hostfwd` assure la redirection des connexions TCP vers le port 22 (SSH) de la machine virtuelle. Le calcul du numéro de port se fait en additionnant 2000 avec le numéro de port SPICE donné lors du lancement du script.

Une fois l'instance de système virtuel lancé, on ouvre une connexion SSH depuis le système hôte en utilisant le port 2010 si on a indiqué 10 lors de l'appel du script.

```
$ ssh -p 2010 etu@localhost
etu@localhost's password:
Linux vm0 3.2.0-4-amd64 #1 SMP Debian 3.2.51-1 x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
No mail.
Last login: Sat Jan  4 16:28:59 2014 from 10.0.2.2
etu@vm0:~$ w
 18:44:18 up  2:13,  1 user,  load average: 0,00, 0,01, 0,05
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
etu      pts/0    10.0.2.2         18:43    2.00s  0.08s  0.00s w
```

Du point de vue configuration réseau, l'interface Ethernet de la machine virtuelle a été configurée via le service DHCP intégré à l'émulateur. La configuration réseau obtenue est la suivante.

```
etu@vm0:~$ ip addr ls
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether ba:ad:00:ca:fe:01 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global eth0
    inet6 fe80::b8ad:ff:feca:fe01/64 scope link
       valid_lft forever preferred_lft forever

etu@vm0:~$ ip route ls
default via 10.0.2.2 dev eth0
10.0.2.0/24 dev eth0  proto kernel  scope link  src 10.0.2.15
```

Les informations relevées dans les copies d'écran ci-dessus montrent que ce sont les fonctions intégrées de l'émulateur qui ont servi à la configuration de l'interface réseau Ethernet de l'hôte virtuel.

9. Fonction TUN/TAP du noyau Linux

La solution retenue pour les communications réseau entre système hôte et machine virtuelle utilise la fonction TUN/TAP du noyau Linux. Cette solution est utilisée dans toutes les sections suivantes de ce document.

Indépendamment du contexte de la virtualisation, `TUN/TAP` est une fonction de réception et de transmission de paquets entre le noyau et les programmes de l'espace utilisateur. Cette fonction peut être vue comme une simple interface point à point ou Ethernet qui, au lieu de recevoir les paquets d'un média physique, les reçoit du programme de l'espace utilisateur. De même, cette interface au lieu d'envoyer les paquets vers un média physique, les transmet au programme de l'espace utilisateur.

Dans le contexte de ce document, le programme de l'espace mémoire utilisateur est l'instance virtuelle de système d'exploitation. L'interface réseau `TUN/TAP` devient un canal de communication réseau entre le système hôte et un système virtualisé. Par analogie, on peut assimiler ce canal de communication à un cordon de brassage qui relie le système de commutation de circuit fourni par le système hôte à l'interface Ethernet du système virtualisé.

La création d'une interface TAP doit se faire par l'intermédiaire d'un programme de l'espace utilisateur. En fonction des usages, plusieurs outils différents permettent de manipuler ces interfaces. On peut citer tunctl qui fait partie du paquet uml-utilities. L'acronyme UML correspond à User Mode Linux une solution qui permet «d'imbriquer» l'exécution de plusieurs systèmes Linux. On peut aussi citer openvpn qui appartient au paquet éponyme. Il s'agit là de manipuler le tunnel correspondant au réseau virtuel.

Cette section se décompose en deux parties consacrées à la mise en œuvre d'une interface TAP. La première propose une configuration d'interface à partir de la session utilisateur alors que la deuxième propose une configuration permanente au niveau système sur la distribution Debian GNU/Linux.

Configuration manuelle d'une interface TAP

Aujourd'hui, la commande ip du paquet iproute2 est devenue le «couteau suisse» des manipulations réseau. C'est elle que l'on utilise le plus souvent.

La création d'une interface utilise la syntaxe suivante :

```
$ sudo❶ ip tuntap add mode tap dev tap1 group kvm multi_queue❷

$ ip addr ls tap1
14: tap1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether f6:98:ad:8f:8e:c1 brd ff:ff:ff:ff:ff:ff

$ sudo ip link set dev tap1 up❸

$ ip addr ls tap1
14: tap1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN group default qlen 1000
link/ether f6:98:ad:8f:8e:c1 brd ff:ff:ff:ff:ff:ff
```

❶ L'utilisation de sudo suppose que les droits aient été délégués au préalable. Soit l'utilisateur a été intégré au groupe système `sudo`, soit il dispose d'une délégation plus précise relative aux outils réseau. À titre d'exemple, le fichier `/etc/sudoers` peut contenir une ligne du type suivant qui autorise tous les membres du groupe `kvm` à utiliser trois commandes utiles :

```
%kvm ALL=(ALL) NOPASSWD: /usr/bin/ovs-vsctl, /sbin/ip, /usr/sbin/iotop
```

❷ L'attribution de l'interface tap1 au groupe système `kvm` est importante. En effet, c'est l'appartenance à ce groupe qui donne accès aux ressources liées à la virtualisation. Dans l'exemple, l'interface créée pourra être utilisée lors de l'initialisation du processus utilisateur correspondant au système virtuel.

L'option `multi_queue` autorise l'utilisation de plusieurs descripteurs de fichiers pour une même interface. Il est ainsi possible de paralléliser l'émission ou la réception de paquets.

❸ Pour être utilisable, l'interface tap1 doit être activée. Pour reprendre l'analogie entre canal de communication mémoire et cordon de brassage, on peut considérer que la prise réseau dispose d'un «interrupteur». La commande d'ouverture de cet interrupteur est symétrique à celle présentée dans la copie d'écran ci-dessus : `$ sudo ip link set dev tap1 down`.

Configuration système d'une interface TAP

Pour généraliser l'utilisation d'une interface `TAP`, il est possible de la paramétrer directement au niveau système dans le fichier de configuration des interfaces réseau. Voici un extrait de fichier `/etc/network/interfaces`.

```
auto tap1
iface tap1 inet manual
        up ip tuntap add mode tap dev $IFACE group kvm multi_queue
        up ip link set dev $IFACE up
        down ip link set dev $IFACE down
        down ip tuntap del mode tap dev $IFACE
```

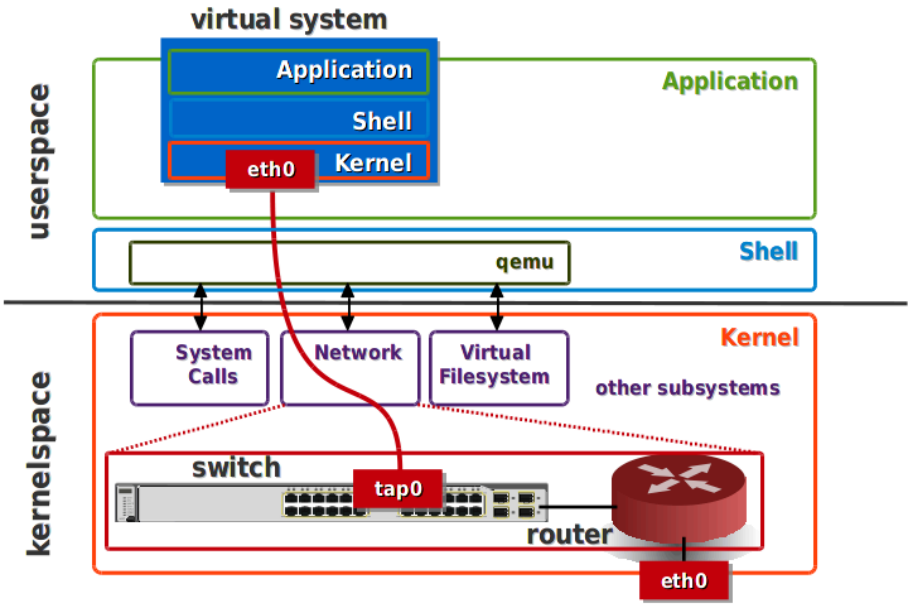
À partir de cet exemple, il est possible de créer autant de «cordons de brassage» que nécessaire sur le système hôte.

Pour conclure cette section, on dispose maintenant d'interfaces qui sont utilisées dans la suite du document pour connecter les instances de systèmes virtuels aux ports du commutateur réseau du système hôte.

10. Communications réseau avec un commutateur virtuel

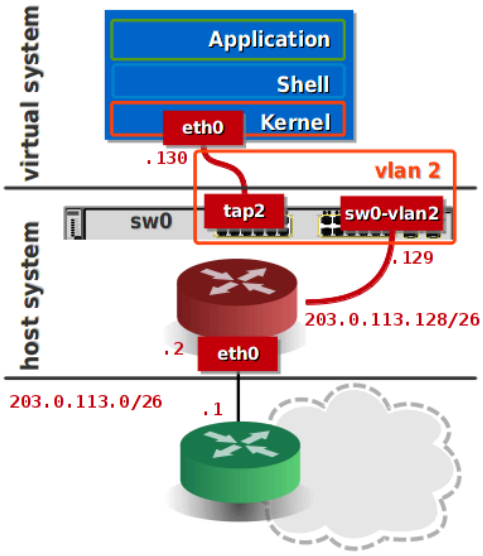
Sur les systèmes GNU/Linux, il existe une grande variété de solutions d'interconnexion réseau entre système hôte et systèmes virtuels. Il se trouve justement que l'interconnexion réseau est la préoccupation principale du site inetdoc. Il est donc essentiel de choisir les outils qui offrent le maximum de fonctionnalités. Le commutateur virtuel occupe une place essentielle dans une solution de virtualisation moderne.

Cette section présente la configuration du commutateur virtuel Open vSwitch. Ce commutateur comprend une partie noyau ainsi que des processus dans l'espace utilisateur. La partie noyau exploite la partie commutation de circuit de celui-ci. Dans le jargon, cette partie est connue sous le nom de Forwarding plane ou encore Forwarding Information Base (FIB). Côté espace utilisateur, on trouve les outils de configuration tels que ovs-vsctl et plusieurs démons de gestion des ports et des VLANs. Il faut ajouter que cet outil exploite les fonctions de routage (commutation de paquets) déjà présentes dans le noyau Linux.



Système hôte & commutation virtuelle - vue complète

Open vSwitch est une pièce maîtresse des architectures d'interconnexion de systèmes virtualisés. Il est présent dans la plupart des solutions intégrées de la famille du Software-defined networking. L'intégration sort du champ couvert par ce guide. On se contente ici de présenter le fonctionnement du commutateur virtuel à l'échelle unitaire. La topologie minimaliste présentée dans la suite correspond à la vue ci-dessous.



Topologie minimum - vue complète

Tableau 1. Plan d'adressage de la topologie étudiée

Système	Interface	Adresse IPv4	Rôle
Machine virtuelle	eth0	`203.0.113.130/26`	Interface réseau de la machine virtuelle
Système hôte	sw0-vlan2	`203.0.113.129/26`	Passerelle par défaut de la machine virtuelle
Système hôte	eth0	`203.0.113.2/26`	Interface réseau du système hôte
Routeur	indéfinie	`203.0.113.1/26`	Passerelle par défaut du système hôte

Installation du commutateur

L'installation du paquet openvswitch-switch comprend la configuration système des outils et des démons de gestion de la configuration du commutateur. Voici quelques éléments relevés après l'installation du paquet.

- Paquets installés

```
$ aptitude search ~iopenvswitch
i A openvswitch-common      - Open vSwitch common components
i   openvswitch-switch      - Open vSwitch switch implementations
```

- Partie noyau

```
$ lsmod | grep openvswitch
openvswitch            63837  0
vxlan                  30915  1 openvswitch
gre                    12957  1 openvswitch
libcrc32c              12426  1 openvswitch
```

- Démons de l'espace utilisateur

```
$ pstree
init─┬─acpid
     ├─atd
     ├─cron
     ├─dbus-daemon
     ├─exim4
     ├─6*[getty]
     ├─ovs-vswitchd───ovs-vswitchd───ovs-vswitchd
     ├─ovsdb-server───ovsdb-server
     ├─rsyslogd───3*[{rsyslogd}]
     └─udevd
```

- État de la configuration

```
$ sudo ovs-vsctl show
cd95ab0d-643d-4a23-bae3-2e6c900a367f
    ovs_version: "1.9.3"
```

Configuration du commutateur

Pour configurer l'interconnexion présentée en début de section, on doit passer par les étapes suivantes.

- Création du commutateur `sw0`

```
$ sudo ovs-vsctl add-br sw0
```

- Création de l'interface réseau pour le VLAN numéro `2`

```
$ sudo ovs-vsctl add-port sw0 sw0-vlan2 tag=2 -- set Interface sw0-vlan2 type=internal
```

- Création du «cordon de brassage» `tap2`

```
$ sudo ip tuntap add mode tap dev tap2 group kvm
```

- Raccordement du «cordon de brassage» `tap2` au commutateur et affectation du port correspondant au VLAN numéro `2`

```
$ sudo ovs-vsctl add-port sw0 tap2 tag=2
```

- Visualisation de la configuration

```
$ sudo ovs-vsctl show
cd95ab0d-643d-4a23-bae3-2e6c900a367f
    Bridge "sw0"
        Port "sw0-vlan2"
            tag: 2
            Interface "sw0-vlan2"
                type: internal
        Port "tap2"
            tag: 2
            Interface "tap2"
        Port "sw0"
            Interface "sw0"
                type: internal
    ovs_version: "1.9.3"
```

Modes accès ou trunk ?

Sur un commutateur Open vSwitch un port est par défaut en mode trunk ; c'est à dire qu'il reçoit le trafic de tous les VLANs «présents sur les autres ports». Dans l'exemple présenté ici, on restreint l'utilisation des ports liés aux interfaces `sw0-vlan2` et `tap2` au seul VLAN numéro 2. On peut donc considérer que ces deux ports sont en mode accès puisqu'ils ne «voient» que le trafic d'un VLAN unique.

Activation des interfaces réseau et du routage

Pour que la configuration du commutateur soit effective, il faut que toutes les nouvelles interfaces réseau soient actives. De plus, on doit s'assurer que la fonction de routage est activée sur le système hôte.

- Activation des interfaces

```
$ for intf in sw0 sw0-vlan2 tap2; do sudo ip link set dev $intf up; done
```

- État des interfaces sur le système hôte

```
$ ip link ls
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN \
                                mode DEFAULT group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP \
                                mode DEFAULT group default qlen 1000
    link/ether ba:ad:00:ca:fe:00 brd ff:ff:ff:ff:ff:ff
4: sw0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN \
                                mode DEFAULT group default
    link/ether 16:8f:91:2c:e0:41 brd ff:ff:ff:ff:ff:ff
5: sw0-vlan2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN \
                                mode DEFAULT group default
    link/ether 8e:2a:a2:ff:d9:fb brd ff:ff:ff:ff:ff:ff
6: tap2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master sw0 state UP \
                                mode DEFAULT group default qlen 500
    link/ether 46:12:48:6b:be:d4 brd ff:ff:ff:ff:ff:ff
```

La fonction de routage fournie par le noyau Linux peut être activée via le fichier `/etc/sysctl.conf`. Il suffit de décommenter les entrées correspondantes. Pour appliquer les paramètres définis dans le fichier on utilise l'instruction `$ sudo sysctl -p`. On peut vérifier les paramètres en n'affichant que les lignes actives de ce même fichier.

```
$ egrep -e '(^$|^#)' -v /etc/sysctl.conf
net.ipv4.ip_forward=1
net.ipv6.conf.all.forwarding=1
net.ipv4.conf.all.log_martians = 1
```

Lancement du système virtuel

Comme dans le cas de la Section 8, « Communications réseau en mode utilisateur », on utilise une script qui rassemble toutes les options de configuration utiles : `ovs-startup.sh`. Le code de ce script est donné en Section A.2, « Communication réseau avec commutateur virtuel Open vSwitch ». La copie d'écran ci-dessous correspond à la topologie minimale décrite plus haut.

```
:~/vm$ ./scripts/ovs-startup.sh vm0-debian-testing-amd64-base.raw 1024 2

~> Machine virtuelle : vm0-debian-testing-amd64-base.raw
~> Port SPICE        : 5902
~> Mémoire RAM       : 1024
~> Adresse MAC       : ba:ad:00:ca:fe:02
```

Les options de ce script relatives aux fonctions réseau sont les suivantes :

```
-device virtio-net❶,netdev=net0❷,mac="$macaddress"❸ \
-netdev tap,ifname=tap$tapnum❹,id=net0,script=no \
```

❶ Le pilote d'interface réseau choisi utilise les fonctions présentées à la Section 4, « KVM et VIRTIO ».

❷ L'identifiant de l'interface utilisé pour le script est `net0`.

❸ L'adresse MAC de l'interface réseau est déterminée à partir du préfixe hexadécimal bad cafe et du numéro de l'interface TAP.

```
macaddress="ba:ad:00:ca:fe:`printf "%02x" $tapnum`"
```

❹ Le raccordement de l'interface réseau identifiée par l'étiquette `net0` se fait via le cordon de brassage (ou l'interface TAP) numéroté aussi avec la variable `$tapnum` du script.

Validation des communications entre système hôte et système virtuel

Pour conclure cette section, on peut valider le lien direct entre les système hôte et virtuel avec une connexion SSH. On commence par identifier l'adresse MAC au format EUI64 du système virtuel puis on l'utilise pour la connexion.

```
$ ip nei ls
fe80::b8ad:ff:feca:fe02 dev sw0-vlan2 lladdr ba:ad:00:ca:fe:02 STALE
fe80::503c:b3ff:febe:b94b dev eth0 lladdr 52:3c:b3:be:b9:4b router REACHABLE

$ ssh etu@fe80::b8ad:ff:feca:fe02%sw0-vlan2
etu@fe80::b8ad:ff:feca:fe02%sw0-vlan2's password:
Linux nested-vm0 3.12-1-amd64 #1 SMP Debian 3.12.6-2 (2013-12-29) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
No mail.
Last login: Thu Jan 23 20:10:50 2014 from fe80::8c2a:a2ff:feff:d9fb%eth0
etu@vm0:~$
```

Vue du système virtuel avec la configuration IP complète, on obtient les informations suivantes.

```
$ ip nei ls
fe80::8c2a:a2ff:feff:d9fb dev eth0 lladdr 8e:2a:a2:ff:d9:fb router DELAY
203.0.113.129 dev eth0 lladdr 8e:2a:a2:ff:d9:fb STALE

$ ip addr ls
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP \
                                              group default qlen 1000
    link/ether ba:ad:00:ca:fe:02 brd ff:ff:ff:ff:ff:ff
    inet 203.0.113.130/26 brd 203.0.113.191 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::b8ad:ff:feca:fe02/64 scope link
       valid_lft forever preferred_lft forever

$ ip ro ls
default via 203.0.113.129 dev eth0
203.0.113.128/26 dev eth0  proto kernel  scope link  src 203.0.113.130
```

11. En guise de conclusion

Ce guide sur l'utilisation des machines virtuelles dans le contexte de l'enseignement est très loin de couvrir tous les usages. L'objectif est de démystifier la virtualisation des fonctions réseau usuelles. D'un côté, on peut affirmer qu'il n'y a rien de nouveau dans la mesure où les fonctions de commutation de trames, de réseaux locaux virtuels (VLAN) et de routage de paquets sont toujours là. D'un autre côté, ces mêmes fonctions se sont déplacées des équipements dédiés vers des systèmes génériques ouvrant le champ à de nouveaux usages : le Software-defined networking.

Du point de vue pédagogique, l'intégration des fonctions réseau classiques, au milieu des autres éléments d'un système, dans des solutions intégrées pose un défi. Doit-on aller de l'utilisation de la solution intégrée vers l'identification des briques fonctionnelles au risque de se «perdre en chemin» ? Doit-on au contraire partir de la modélisation réseau et donc des fonction élémentaires pour «construire» la solution intégrée ? Ce guide, dans son mode de rédaction, penche résolument pour la seconde solution. L'avenir nous dira si c'est pertinent.

A. Scripts spécifiques

Voici un rappel des codes des différents scripts utilisés dans ce document.

A.1. Communications réseau en mode utilsateur

Le script `standalone-startup.sh` sert à lancer une instance de système virtualisé dont la configuration réseau est prédéfinie en mode utilisateur. Ce mode est présenté à la Section 8, « Communications réseau en mode utilisateur ».

Accès : standalone-startup.sh

```
#!/bin/bash

RedOnBlack='\E[31m'
```

```
vm=$1
shift
memory=$1
shift
tapnum=$1
shift

if [[ -z "$vm" || -z "$memory" || -z "$tapnum" ]]
then
 echo "ERREUR : paramètre manquant"
 echo "Utilisation : $0 <fichier image> <quantité mémoire en Mo> <numéro de port SPICE>"
 exit 1
fi

if (( $memory < 128 ))
then
 echo "ERREUR : quantité de mémoire RAM insuffisante"
 echo "La quantité de mémoire en Mo doit être supérieure ou égale à 128"
 exit 1
fi

second_rightmost_byte=$(printf "%02x" $(expr $tapnum / 256))
rightmost_byte=$(printf "%02x" $(expr $tapnum % 256))
macaddress="ba:ad:ca:fe:$second_rightmost_byte:$rightmost_byte"

image_format="${vm##*.}"

echo -e "$RedOnBlack"
echo "~> Machine virtuelle : $vm"
echo "~> Port SPICE        : $((5900 + $tapnum))"
echo "~> Mémoire RAM       : $memory"
echo "~> Adresse MAC       : $macaddress"
tput sgr0

ionice -c3 qemu-system-x86_64 \
 -machine type=q35,accel=kvm:tcg \
 -cpu max \
 -device intel-iommu \
 -daemonize \
 -name $vm \
 -m $memory \
 -device virtio-balloon \
 -smp 2,threads=2 \
 -rtc base=localtime,clock=host \
 -watchdog i6300esb \
 -watchdog-action none \
 -boot order=c,menu=on \
 -object "iothread,id=iothread.drive0" \
 -drive if=none,id=drive0,aio=native,cache.direct=on,discard=unmap,format=$image_format,media=disk,file=$vm \
 -device virtio-blk,num-queues=4,drive=drive0,scsi=off,config-wce=off,iothread=iothread.drive0 \
 -k fr \
 -vga qxl \
 -spice port=$((5900 + $tapnum)),addr=localhost,disable-ticketing \
 -device virtio-serial-pci \
 -device virtserialport,chardev=spicechannel0,name=com.redhat.spice.0 \
 -chardev spicevmc,id=spicechannel0,name=vdagent \
 -usb \
 -device usb-tablet,bus=usb-bus.0 \
 -device intel-hda \
 -device hda-duplex \
 -device virtio-net,netdev=net0,mac="$macaddress" \
 -netdev user,id=net0,hostfwd=tcp::$((2200 + $tapnum))-:22 \
 $*
```

A.2. Communication réseau avec commutateur virtuel Open vSwitch

Le script `ovs-startup.sh` sert à lancer une instance de système virtualisé raccordé à un commutateur virtuel via une interface TAP (Voir Section 9, « Fonction TUN/TAP du noyau Linux »). Ce mode de fonctionnement est présenté à la Section 10, « Communications réseau avec un commutateur virtuel ».

Accès : ovs-startup.sh

```
#!/bin/bash

# This script is part of https://inetdoc.net project
#
# It starts a qemu/kvm x86 virtual machine plugged into an Open VSwitch port
# through an already existing tap interface.
# It should be run by a normal user account which belongs to the kvm system
```

```
# group and is able to run the ovs-vsctl command via sudo
#
# This version of the virtual machine startup script uses the UEFI boot sequence
# based on the files provided by the ovmf package.
# The qemu parameters used here come from ovml package readme file
# Source: https://github.com/tianocore/edk2/blob/master/OvmfPkg/README
#
# File: ovs-startup.sh
# Author: Philippe Latu
# Source: https://github.com/platu/inetdoc/blob/master/guides/vm/files/ovs-startup.sh
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>.

RED='\e[1;31m'
GREEN='\e[1;32m'
BLUE='\e[1;34m'
NC='\e[0m' # No Color

vm=$1
shift
memory=$1
shift
tapnum=$1
shift

# Are the 3 parameters there ?
if [[ -z "${vm}" || -z "${memory}" || -z "${tapnum}" ]]
then
 echo -e "${RED}ERROR : missing parameter.${NC}"
 echo -e "${GREEN}Usage : $0 [image file] [RAM size in MB] [tap interface number]${NC}"
 exit 1
fi

# Does the VM image file exist ?
if [[ ! -f "${vm}" ]]
then
 echo -e "${RED}ERROR : the ${vm} image file does not exist.${NC}"
 exit 1
fi

# Is the amount of ram sufficient to run the VM ?
if [[ ${memory} -lt 128 ]]
then
 echo -e "${RED}ERROR : unsifficient RAM size : ${memory}MB${NC}"
 echo -e "${GREEN}RAM size must be above 128MB.${NC}"
 exit 1
fi

# Is the tap interface free ?
if [[ ! -z "$(ps aux | grep =[t]ap${tapnum}, )" ]]
then
 echo -e "${RED}tap${tapnum} is already in use by another process.${NC}"
 exit 1
fi

# Are the OVMF symlink and file copy there ?
if [[ ! -L "./OVMF_CODE.fd" ]]
then
 ln -s /usr/share/OVMF/OVMF_CODE.fd .
fi

if [[ ! -f "${vm}_OVMF_VARS.fd" ]]
then
 cp /usr/share/OVMF/OVMF_VARS.fd ${vm}_OVMF_VARS.fd
fi

# Is it possible to set a new Software TPM socket ?
if [[ -z "$(which swtpm)" ]]
then
 echo -e "${RED}TPM emulator not available${NC}"
```

```
 exit 1
fi

# Does the software TPM directory exists ?
tpm_dir=${vm}_TPM

if [[ ! -d "${tpm_dir}" ]]
then
 mkdir ${tpm_dir}
fi

# Is swtpm already there for this virtual machine
tpm_pid=$(pgrep -u $USER -a swtpm | grep ${tpm_dir}/swtpm-sock | cut -f 1 -d ' ')
if [[ ! -z "${tpm_pid}" ]]
then
 kill ${tpm_pid}
fi

swtpm socket \
 --tpmstate dir=${tpm_dir} \
 --ctrl type=unixio,path=${tpm_dir}/swtpm-sock \
 --log file=${tpm_dir}/swtpm.log \
 --tpm2 \
 --terminate &

# Is the switch port available ? Which mode ? Which VLAN ?
second_rightmost_byte=$(printf "%02x" $(expr ${tapnum} / 256))
rightmost_byte=$(printf "%02x" $(expr ${tapnum} % 256))
macaddress="b8:ad:ca:fe:$second_rightmost_byte:$rightmost_byte"
lladdress="fe80::baad:caff:fefe:$(printf "%x" ${tapnum})"
vlan_mode="$(sudo ovs-vsctl list port tap${tapnum} | grep vlan_mode | egrep -o '(access|trunk)')"

if [[ "$vlan_mode" == "access" ]]
then
 svi="vlan$(sudo ovs-vsctl list port tap${tapnum} | grep tag | grep -o -E '[0-9]+')"
else
 svi="dsw-host"
fi

image_format="${vm##*.}"

spice=$((5900 + ${tapnum}))
telnet=$((2300 + ${tapnum}))

# Is TPM socket is ready.
wait=0

while [[ ! -S ${tpm_dir}/swtpm-sock ]] && [[ $wait -lt 10 ]]
do
 echo "Waiting a second for TPM socket to be ready."
 sleep 1s
 ((wait++))
done

if [[ ${wait} -eq 10 ]]
then
 echo -e "${RED}TPM socket setup failed. Giving up.${NC}"
 exit 1
fi

echo -e "~> Virtual machine filename   : ${RED}${vm}${NC}"
echo -e "~> RAM size                   : ${RED}${memory}MB${NC}"
echo -e "~> SPICE VDI port number      : ${GREEN}${spice}${NC}"
echo -e "~> telnet console port number : ${GREEN}${telnet}${NC}"
echo -e "~> MAC address                : ${BLUE}${macaddress}${NC}"
echo -e "~> Switch port interface      : ${BLUE}tap${tapnum}, ${vlan_mode} mode${NC}"
echo -e "~> IPv6 LL address            : ${BLUE}${lladdress}%${svi}${NC}"
tput sgr0

ionice -c3 qemu-system-x86_64 \
 -machine type=q35,accel=kvm:tcg,kernel-irqchip=split \
 -cpu max,l3-cache=on,+vmx \
 -device intel-iommu,intremap=on \
 -daemonize \
 -name ${vm} \
 -m ${memory} \
 -device virtio-net-pci,mq=on,vectors=6,netdev=net${tapnum},disable-legacy=on,disable-modern=off,mac="${macaddress}
 -netdev tap,queues=2,ifname=tap${tapnum},id=net${tapnum},script=no,downscript=no,vhost=on \
 -serial telnet:localhost:${telnet},server,nowait \
 -device virtio-balloon \
 -smp 8,threads=2 \
```

```
 -rtc base=localtime,clock=host \
 -device i6300esb \
 -watchdog-action poweroff \
 -boot order=c,menu=on \
 -object iothread,id=iothread.drive0 \
 -drive if=none,id=drive0,aio=native,cache.direct=on,discard=unmap,format=${image_format},media=disk,l2-cache-size=
 -device virtio-blk,num-queues=4,drive=drive0,scsi=off,config-wce=off,iothread=iothread.drive0 \
 -global driver=cfi.pflash01,property=secure,value=on \
 -drive if=pflash,format=raw,unit=0,file=OVMF_CODE.fd,readonly=on \
 -drive if=pflash,format=raw,unit=1,file=${vm}_OVMF_VARS.fd \
 -k fr \
 -vga none \
 -device qxl-vga,vgamem_mb=64 \
 -spice port=${spice},addr=localhost,disable-ticketing=on \
 -device virtio-serial-pci \
 -device virtserialport,chardev=spicechannel0,name=com.redhat.spice.0 \
 -chardev spicevmc,id=spicechannel0,name=vdagent \
 -object rng-random,filename=/dev/urandom,id=rng0 \
 -device virtio-rng-pci,rng=rng0 \
 -chardev socket,id=chrtpm,path=${tpm_dir}/swtpm-sock \
 -tpmdev emulator,id=tpm0,chardev=chrtpm \
 -device tpm-tis,tpmdev=tpm0 \
 -usb \
 -device usb-tablet,bus=usb-bus.0 \
 -device ich9-intel-hda,addr=1f.1 \
 -audiodev spice,id=snd0 \
 -device hda-output,audiodev=snd0 \
 $*
```

A.3. Gestion des images différentielles

Le script `diff-img.sh` permet de créer une image différentielle relativement à une première image dite master. L'intérêt de cette opération est de pouvoir lancer une instance de système virtuel «jetable». En effet, on peut considérer que toutes les opérations effectuées sur l'image différentielle n'ont pas à être conservées. Dans ce cas, il suffit d'effacer le fichier d'image différentielle après usage.

Accès : diff-img.sh

```
#!/bin/bash

image_format="${1##*.}"

if [[ -z "$1" || -z "$2" ]]
then
 echo "ERREUR : paramètre manquant"
 echo "Utilisation : $0 <fichier image source> <fichier image différentiel>"
 exit 1
fi

qemu-img create -b $1 -f $image_format $2 >/dev/null

exit 0
```

A.4. Installation d'un système sur machine virtuelle

Le script `iso-install-startup.sh` reprend le code du Section A.1, « Communications réseau en mode utilsateur » avec la description d'un lecteur de DVD pour que l'on puisse lancer une installation système sur une image vierge. Ce script n'est utile que pour la création d'un nouveau master.

Accès : iso-install-startup.sh

```
#!/bin/bash

# This script is part of https://inetdoc.net project
#
# It starts a qemu/kvm x86 virtual machine plugged into an Open VSwitch port
# through an already existing tap interface.
# It should be run by a normal user account which belongs to the kvm system
# group and is able to run the ovs-vsctl command via sudo
#
# This version of the virtual machine startup script uses the UEFI boot sequence
# based on the files provided by the ovmf package.
# The qemu parameters used here come from ovml package readme file
# Source: https://github.com/tianocore/edk2/blob/master/OvmfPkg/README
#
# File: iso-install-startup.sh
# Author: Philippe Latu
```

```
# Source: https://github.com/platu/inetdoc/blob/master/guides/vm/files/iso-install-startup.sh
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>.

RED='\e[1;31m'
GREEN='\e[1;32m'
BLUE='\e[1;34m'
NC='\e[0m' # No Color

vm=$1
shift
iso=$1
shift
memory=$1
shift
tapnum=$1
shift

# Are the 4 parameters there ?
if [[ -z "${vm}" || -z "${iso}" || -z "${memory}" || -z "${tapnum}" ]]
then
 echo -e "${RED}ERROR : missing parameter.${NC}"
 echo -e "${GREEN}Usage : $0 [image file] [iso file] [RAM size in MB] [tap interface number]${NC}"
 exit 1
fi

# Does the VM image file exist ?
if [[ ! -f "${vm}" ]]
then
 echo -e "${RED}ERROR : the ${vm} image file does not exist.${NC}"
 exit 1
fi

# Is the amount of ram sufficient to run the VM ?
if [[ ${memory} -lt 128 ]]
then
 echo -e "${RED}ERROR : unsifficient RAM size : ${memory}MB${NC}"
 echo -e "${GREEN}RAM size must be above 128MB.${NC}"
 exit 1
fi

# Is the tap interface free ?
if [[ ! -z "$(ps aux | grep =[t]ap${tapnum}, )" ]]
then
 echo -e "${RED}tap${tapnum} is already in use by another process.${NC}"
 exit 1
fi

# Are the OVMF symlink and file copy there ?
if [[ ! -L "./OVMF_CODE.fd" ]]
then
 ln -s /usr/share/OVMF/OVMF_CODE.fd .
fi

if [[ ! -f "${vm}_OVMF_VARS.fd" ]]
then
 cp /usr/share/OVMF/OVMF_VARS.fd ${vm}_OVMF_VARS.fd
fi

# Is it possible to set a new Software TPM socket ?
if [[ -z "$(which swtpm)" ]]
then
 echo -e "${RED}TPM emulator not available${NC}"
 exit 1
fi

# Does the software TPM directory exists ?
tpm_dir=${vm}_TPM

if [[ ! -d "${tpm_dir}" ]]
```

```
then
 mkdir ${tpm_dir}
fi

# Is swtpm already there for this virtual machine
tpm_pid=$(pgrep -u $USER -a swtpm | grep ${tpm_dir}/swtpm-sock | cut -f 1 -d ' ')
if [[ ! -z "${tpm_pid}" ]]
then
 kill ${tpm_pid}
fi

swtpm socket \
 --tpmstate dir=${tpm_dir} \
 --ctrl type=unixio,path=${tpm_dir}/swtpm-sock \
 --log file=${tpm_dir}/swtpm.log \
 --tpm2 \
 --terminate &

# Is the switch port available ? Which mode ? Which VLAN ?
second_rightmost_byte=$(printf "%02x" $(expr ${tapnum} / 256))
rightmost_byte=$(printf "%02x" $(expr ${tapnum} % 256))
macaddress="b8:ad:ca:fe:$second_rightmost_byte:$rightmost_byte"
lladdress="fe80::baad:caff:fefe:$(printf "%x" ${tapnum})"
vlan_mode="$(sudo ovs-vsctl list port tap${tapnum} | grep vlan_mode | egrep -o '(access|trunk)')"

if [[ "$vlan_mode" == "access" ]]
then
 svi="vlan$(sudo ovs-vsctl list port tap${tapnum} | grep tag | grep -o -E '[0-9]+')"
else
 svi="dsw-host"
fi

image_format="${vm##*.}"

spice=$((5900 + ${tapnum}))
telnet=$((2300 + ${tapnum}))

# Is TPM socket is ready.
if [[ ! -S ${tpm_dir}/swtpm-sock ]]
then
 echo "Waiting a second for TPM socket to be ready."
 sleep 1s
fi

echo -e "~> Virtual machine filename   : ${RED}${vm}${NC}"
echo -e "~> RAM size                   : ${RED}${memory}MB${NC}"
echo -e "~> SPICE VDI port number      : ${GREEN}${spice}${NC}"
echo -e "~> telnet console port number : ${GREEN}${telnet}${NC}"
echo -e "~> MAC address                : ${BLUE}${macaddress}${NC}"
echo -e "~> Switch port interface      : ${BLUE}tap${tapnum}, ${vlan_mode} mode${NC}"
echo -e "~> IPv6 LL address            : ${BLUE}${lladdress}%${svi}${NC}"
tput sgr0

ionice -c3 qemu-system-x86_64 \
 -machine type=q35,accel=kvm:tcg \
 -cpu max,l3-cache=on,+vmx \
 -device intel-iommu \
 -daemonize \
 -name ${vm} \
 -m ${memory} \
 -device virtio-net-pci,mq=on,vectors=6,netdev=net${tapnum},disable-legacy=on,disable-modern=off,mac="${macaddress}
 -netdev tap,queues=2,ifname=tap${tapnum},id=net${tapnum},script=no,downscript=no,vhost=on \
 -serial telnet:localhost:${telnet},server,nowait \
 -device virtio-balloon \
 -smp 8,threads=2 \
 -rtc base=localtime,clock=host \
 -device i6300esb \
 -watchdog-action poweroff \
 -boot once=d,menu=on \
 -device ahci,id=ahci0 \
 -device ide-cd,bus=ahci0.0,drive=drive-sata0-0-0,id=sata0-0-0 \
 -drive media=cdrom,if=none,file=$iso,id=drive-sata0-0-0 \
 -object "iothread,id=iothread.drive0" \
 -drive if=none,id=drive0,aio=native,cache.direct=on,discard=unmap,format=${image_format},media=disk,file=${vm} \
 -device virtio-blk,num-queues=4,drive=drive0,scsi=off,config-wce=off,iothread=iothread.drive0 \
 -global driver=cfi.pflash01,property=secure,value=on \
 -drive if=pflash,format=raw,unit=0,file=OVMF_CODE.fd,readonly=on \
 -drive if=pflash,format=raw,unit=1,file=${vm}_OVMF_VARS.fd \
 -k fr \
 -vga none \
 -device qxl-vga,vgamem_mb=64 \
```

```
-spice port=${spice},addr=localhost,disable-ticketing=on \
-device virtio-serial-pci \
-device virtserialport,chardev=spicechannel0,name=com.redhat.spice.0 \
-chardev spicevmc,id=spicechannel0,name=vdagent \
-object rng-random,filename=/dev/urandom,id=rng0 \
-device virtio-rng-pci,rng=rng0 \
-chardev socket,id=chrtpm,path=${tpm_dir}/swtpm-sock \
-tpmdev emulator,id=tpm0,chardev=chrtpm \
-device tpm-tis,tpmdev=tpm0 \
-usb \
-device usb-tablet,bus=usb-bus.0 \
-device ich9-intel-hda,addr=1f.1 \
-audiodev spice,id=snd0 \
-device hda-output,audiodev=snd0 \
$*
```