Nouvel Hyperviseur

Table of Contents

Préparation du système de base

Configuration du réseau

Définir l'adresse de l'interface ipv4 réseau :

```
nmcli conn mod ens1f0 ipv4.method manual ipv4.addresses 10.13.254.153/24
ipv4.gateway 10.13.254.7 ipv4.dns "10.154.59.104,10.156.32.33"
```

Préparation du système

Mettre à jour les paquets

```
dnf -y update
```

Installer les paquets minimum requis

Les dépôts **epel** doivent être activés pour pouvoir installer **dkms**:

`yum install epel-release`

```
dnf install -y dkms git curl wget vim kernel-devel libguestfs libguestfs-
tools
```

Installation de Btrfs

Rocky8 a abandonné le support du système de fichier Btrfs, pour l'implémenter il faut compiler les deu deux parties:

- **btrfs.ko**: Module noyau (pilote) pour Linux. Généralement situé dans /lib/modules/4.2.0/kernel/fs/btrfs/btrfs.ko et doit être pris en charge par le noyau Linux.
- **brtfs-tools**: Les sources des utilitaires de l'espace utilisateur.

Pour disposer d'une toolchain afin de compiler les sources sans polluer le système de fichier du serveur on va utiliser un conteneur **nspawn**

```
yum -y --releasever=8 --nogpg --installroot=/data/nfsshare/toolchain/gcc-el8
--disablerepo='*' --enablerepo=baseos --enablerepo=devel --
enablerepo=appstream install systemd passwd yum rocky-release adobe-
mappings-cmap adobe-mappings-cmap-deprecated adobe-mappings-pdf asciidoc atk
autoconf automake boost-filesystem boost-timer byacc copy-jdk-configs ctags
diffstat docbook-dtds docbook-style-xsl dyninst elfutils elfutils-
debuginfod-client-devel elfutils-devel gc gcc-c++ gcc-gdb-plugin gdb gdb-
headless gdk-pixbuf2-modules gettext-common-devel gettext-devel google-
droid-sans-fonts graphviz gtk-update-icon-cache gtk2 guile hicolor-icon-
theme intltool jasper-libs java javapackages-filesystem jbig2dec-libs jna
lcms2 libICE libSM libXaw libXcomposite libXcursor libXdamage libXi
libXinerama libXmu libXrandr libXt libXxf86misc libatomic_ops libbabeltrace
libfontenc libgs libidn libijs libipt libmcpp libpaper librsvg2 libstdc++-
devel libtool libtool-ltdl libzstd-devel lksctp-tools ltrace lua mcpp nss-
tools openjpeg2 patch patchutils perl-Fedora-VSP perl-Sys-Syslog perl-
Thread-Queue perl-XML-Parser perl-generators pesign rpm-build rpm-sign sgml-
common source-highlight systemtap systemtap-client systemtap-devel
```

```
systemtap-runtime tbb tzdata-java urw-base35-bookman-fonts urw-base35-c059-
fonts urw-base35-d050000l-fonts urw-base35-fonts urw-base35-fonts-common
urw-base35-gothic-fonts urw-base35-nimbus-mono-ps-fonts urw-base35-nimbus-
roman-fonts urw-base35-nimbus-sans-fonts urw-base35-p052-fonts urw-base35-
standard-symbols-ps-fonts urw-base35-z003-fonts valgrind valgrind-devel
xorg-x11-font-utils xorg-x11-fonts-ISO8859-1-100dpi xorg-x11-server-utils
xz-devel zstd cmake cmake-data cmake-filesystem cmake-rpm-macros fuse fuse-
common hexedit hivex libgcrypt-devel libgpg-error-devel libguestfs
libguestfs-appliance libguestfs-tools-c libguestfs-xfs libuv libvirt-daemon-
kvm libxml2-devel libxslt-devel mtools ruby ruby-devel ruby-irb ruby-libs
rubygem-bigdecimal rubygem-did_you_mean rubygem-io-console rubygem-json
rubygem-openssl rubygem-psych rubygem-rdoc rubygems scrub supermin syslinux
syslinux-extlinux syslinux-extlinux-nonlinux syslinux-
nonlinux-6.04-6.el8.noarch libxslt-devel libxml2-devel libvirt-devel
libguestfs-tools-c ruby-devel
```

Sur une distribution Rocky Linux, on peut rapidement installer la plupart des outils de développement nécessaires en exécutant cette commande :

```
dnf -y groupinstall 'Developpement tools'
```

Pour une raison quelconque, dnf n'a pas défini la “bonne” étiquette SELinux sur /etc/passwd. Mais il a mis une étiquette sur `/bin/passwd`. Cette incompatibilité cause un problème pour le changement du mot de passe root. Tenter d'exécuter `restorecon -Rv /` à l'intérieur du conteneur ne fait rien car IIRC libselinux détecte quand il est exécuté dans un conteneur et ne fait rien. Il faut donc l'exécuter depuis l'extérieur du conteneur:

```
restorecon -Rv /data/nfsshare/toolchain/gcc-el8
```

Entrer dans le conteneur pour configurer le mot de passe root:

```
systemd-nspawn -D /data/nfsshare/toolchain/gcc-el8
```

Puis démarrer le conteneur:

```
systemd-nspawn -bD /data/nfsshare/toolchain/gcc-el8
```

Pour compiler le source de btrfs-tools il faut installer les paquets suivants

```
dnf install git asciidoc xmlto
dnf --enablerepo=powertools --enablerepo=devel install python3-sphinx
```

```
e2fsprogs-devel libblkid-devel libudev-devel python3-devel lzo-devel
```

Récupérer la source du paquet **Btrfs-tools** et construire l'application:

```
git config --global http.proxy http://proxy.infra.dgfip:3128
git clone
https://git.kernel.org/pub/scm/linux/kernel/git/kdave/btrfs-progs.git
cd btrfs-progs/
./autogen.sh
./configure
cp Documentation/index.rst Documentation/contents.rst
make
make DESTDIR=$PWD/_pkg install
tar -C _pkg/ -cvf ../btrfs-progs-el8.tar.gz ./
```

Pour installer le paquet il suffit de décompresser l'archive à la racine du système hôte:

```
tar xf /data/nfsshare/toolchain/gcc-el8/root/btrfs-progs-el8.tar.gz -C /
```

Pour le module **btrfs.ko** il ne suffit pas de compiler le module mais il faut recompiler le noyau car celui-ci doit exporter des fonctionnalités dont le module a besoin.

Installer les bibliothèques, fichiers d'en-tête et applications dont on peut avoir besoin:

```
dnf -y install ncurses-devel openssl-devel elfutils-libelf-devel python3 bc
```

Enfin, installer le package **dwarves** à partir du référentiel **Powertools**:

```
dnf -y --enablerepo=powertools install dwarves
```

Pour connaître la version du noyau installé utiliser la commande suivante:

```
uname -r 4.18.0-477.15.1.el8_8.x86_64
```

La version du noyau que l'on va utiliser est la version 4.18.0 qui est disponible sur `www.kernel.org/pub/linux/kernel/v4.x/`:

echo "*%topdir /root/rpm-src/kernel4-18-0" > ~/.rpmmacros rpm -ivh kernel-4.18.0-477.15.1.el8*8.src.rpm

```
wget https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.18.tar.gz
```

Le noyau est disponible dans le rpm du kernel linux dans les dépôts des sources:
Pour récupérer celui-ci:

- Télécharger le paquet source du kernel installé: `yum download --source kernel`
- Indiquer au système où placer les sources: `echo "%_topdir /root/rpm-src/kernel-4.18.0" > ~/.rpmmacros`
- Extraire les sources dans le dossier spécifié: `rpm -ivh kernel-4.18.0-477.15.1.el8_8.src.rpm`

On trouvera dans le dossier indiqué un dossier SOURCES contant le paquet t le fichier config utilisé pour construire le noyau

Décompresser le tarball du noyau:.

```
tar xvf linux-4.18.20.tar.xz
```

Deux étapes principales sont nécessaires à la construction d'un noyau :

- configuration
- compiler

La première étape de la construction du noyau consiste à configurer ses fonctionnalités.

Préparer le dossier des sources pour construire le kernel:

```
cd ~/linux-4.18.20
make  mrproper
```

En dehors du conteneur de toolchain sur le système d'origine recopier le fichier de configuration issu de l'installation le fichier de configuration:

```
cp ~/rpm-src/kernel-4.18.0/SOURCES/kernel-x86_64.config
~/linux-4.18.20/.config
```

Vazlider la configuration ainsi importée:

```
make menuconfig
```

En l'absence de clès pem valides pour signer les modules du kernel il faut déconfigurer **CONFIG_SYSTEM_TRUSTED_KEYS** en utilisant la commande sed suivante:

```
sed -ri '/CONFIG_SYSTEM_TRUSTED_KEYS/s/=.+/=""/g' ~/build/kernel/.config
```

Utiliser la commande sed suivante afin de personnaliser la version du noyau:

```
sed -i 's/^EXTRAVERSION.*/EXTRAVERSION = -custom/' Makefile
```

Le module **ZFS** n'est pas dans la configuration par défaut il faut donc l'ajouter en utilisant les sources.

Récupérer les sources et préparer un dossier temporaire pour le noyau:

```
mkdir ~/src && cd ~/src
git clone -b linux-4.18.20
git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-
stable.git
git clone https://github.com/zfsonlinux/zfs
cd ~/src/linux-4.18.20
make prepare
```

Configurer et ajouter ZFS au noyau:

```
cd ~/src/zfs
git checkout master
sh autogen.sh
./configure --prefix=/ --libdir=/lib --includedir=/usr/include --
datarootdir=/usr/share --enable-linux-builtin=yes --with-
linux=/root/src/linux-4.18.20 --with-linux-
obj=/root/src/linux-4.18.20
```

Intégrer le module dans le répertoire propre du kernel
`./copy-builtin ~/linux-4.18.20`

Configurer le kernel pour y ajouter **zfs** et **btrfs**:

```
make menuconfig
```

Compiler le kernel

```
~/linux-4.18.20
make
```

Les étapes suivantes doivent être réalisées sur le système cible (en dehors du toolchain gcc-el8.

Comme on a compilé des parties du noyau sous forme de modules, il faut installer les modules:

```
cd /data/nfsshare/toolchain/gcc-el8/root/linux-4.18.20
```

```
make INSTALL_MOD_STRIP=1 modules_install
```

Placer le nouveau noyau dans le répertoire `/boot`

```
cp /data/nfsshare/toolchain/gcc-el8/root/linux-4.18.20/arch/x86/boot/bzImage
/boot/vmlinuz-4.18.20-custom
cp -v  /data/nfsshare/toolchain/gcc-el8/root/linux-4.18.20/System.map
/boot/System.map-4.18.20-custom
```

La commande **kernel-install** permet d'ajouter automatiquement à GRUB2 une nouvelle entrée:

```
kernel-install add  4.18.20-custom /boot/vmlinuz-4.18.20-custom
```

Redémarrer le système.

Installation de ZFS

Installer zfs

```
rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-zfsonlinux
dnf install https://zfsonlinux.org/epel/zfs-release-2-2$(rpm --eval
"%{dist}").noarch.rpm
dnf install zfs
```

zfs nécessite l'installations de 210 paquets

La vérification par l'antivirus ne permet pas le téléchargement des modules par http avec des performances suffisantes. Il est donc préférable de forcer l'installation par https: `dnf install https://mirror.hoztnode.net/zfs/epel/9/x86_64/zfs-2.1.12-1.el9.x86_64.rpm`

Vérifier que l'installation du module est correcte

```
dkms status
zfs/2.1.12, 5.14.0-284.18.1.el9_2.x86_64, x86_64: installed
```

Configuration de l'espace de stockage

Intégrer le volume des données dans le pv lvm

```
pvcreate /dev/sdb1
```

Créer le volume group vg1

```
vgcreate vg1 /dev/sdb1
```

Créer les volumes logiques:

```
lvcreate -n vol1 -L 1100g vg1
lvcreate -n vol2 -L 1100g vg1
lvcreate -n vol3 -L 1100g vg1
```

Créer les systèmes de fichiers:

```
mkfs -t xfs /dev/vg1/vol1
mkfs -t xfs /dev/vg1/vol2
```

Créer un pool zfs où stocker les containers:

```
zpool create zpool_lxd /dev/vg1/vol3 -f
```

Créer les dossiers de montage:

```
mkdir -pv /data/{libvirt,container,nfsshare}
```

Ajout des points de montage dans `/etc/fstab`:

```
/dev/vg1/vol1           /data/libvirt           xfs     defaults        0 0
/dev/vg1/vol2           /data/nfsshare          xfs     defaults        0 0
```

Rechargement du démon et montage manuel des systèmes de fichiers:

```
systemctl daemon-reload
mount /data/libvirt
mount /data/nfsshare
mount /data/container
```

Monter le pool zfs:

```
sudo zfs set mountpoint=/data/container zpool_lxd
```

Vérifier le montage correcte des systèmes de fichiers:

```
df -h
/dev/mapper/vg1-vol1   1,1T    7,7G  1,1T   1% /data/libvirt
/dev/mapper/vg1-vol2   1,1T    7,7G  1,1T   1% /data/nfsshare
zpool_lxd              1,1T    128K  1,1T   1% /data/container
```

KVM

Prérequis

Vérifier si le processeur supporte la virtualisation matérielle :

```
grep -E 'svm|vmx' /proc/cpuinfo

flags : ... vmx ...
```

Installation

KVM et les outils correspondants sont fournis par les dépôts officiels de la distribution :

```
dnf install -y qemu-kvm libvirt virt-install bridge-utils
```

Mise en service

Vérifier si les modules KVM sont chargés :

```
lsmod | grep kvm
kvm_intel             344064  0
kvm                   958464  1 kvm_intel
irqbypass              16384  1 kvm
```

Lancer les services **libvirtd** et **libvirt-guests** :

```
systemctl enable libvirtd --now
systemctl enable libvirt-guests --now
```

Socket TCP

Masquer tous les fichiers socket unit pour revenir au mode traditionnel

```
systemctl mask libvirtd.socket libvirtd-ro.socket \
   libvirtd-admin.socket libvirtd-tls.socket libvirtd-tcp.socket
```

Créer l'espace de stockage de ces certificats:

```
mkdir -pv /etc/pki/{CA,libvirt/private}
```

Créer un Certificate Authority (CA):

```
certtool --generate-privkey > cakey.pem

cat > ca.info <<EOF
cn DGFIP sns, Inc.
ca
cert_signing_key
EOF

certtool --generate-self-signed --load-privkey cakey.pem \
  --template ca.info --outfile cacert.pem
cp cacert.pem /etc/pki/CA/
```

Créer une clé privée pour le serveur :

```
certtool --generate-privkey > serverkey.pem
```

et signer cette clé avec la clé privée de l'autorité de certification en créant d'abord un fichier modèle appelé server.info. Le fichier de modèle contiendra un certain nombre de champs pour définir le serveur comme suit :

```
cat > server.info<< 'EOF'
organization = dgfip
cn = compute1.libvirt.org
ip_address = 10.13.254.153
tls_www_server
encryption_key
signing_key
EOF
```

Utiliser le fichier de modèle comme entrée d'une commande certtool pour signer le certificat du serveur :

```
certtool --generate-certificate --load-privkey serverkey.pem \
  --load-ca-certificate cacert.pem --load-ca-privkey cakey.pem \
  --template server.info --outfile servercert.pem

mkdir -pv pki/libvirt/private/
```

```
cp serverkey.pem /etc/pki/libvirt/private/
cp servercert.pem /etc/pki/libvirt/
```

Pour activer le mode traditionnel:

- activer l'écoute libvirtd: `echo 'LIBVIRTD_ARGS="--listen"' >/etc/sysconfig/libvirtd`
- modifier le fichier `/etc/libvirt/libvirtd.conf` pour décommenter la ligne **listen_tcp**:`sed -i -e "s/#listen_tcp = 1/listen_tcp = 1/g" /etc/libvirt/libvirtd.conf`

Pour utiliser la connection avec mot de passe:
* installer le plugin digest-md5: `dnf install -y cyrus-sasl-md5`
* remplacer dans le fichier `/etc/sasl2/libvirt.conf` le mécanisme **gssapi** par **digest-md5**: `mech_list: digest-md5`
* décommenter dans le fichier `/etc/sasl2/libvirt.conf` la ligne contenant **sasldb_path**: `sasldb_path: /etc/libvirt/passwd.db`
* initialiser la base de données passwd.db: `saslpasswd2 -a libvirtd -f /etc/libvirt/passwd.db admin`

Ouvrir l'accès des ports libvirtd dans le Firewall

```
firewall-cmd --permanent --add-port=16509/tcp
firewall-cmd --reload
```

Redémarrer linvirtd

```
systemctl restart libvirtd
```

Kerberos

Afin permettre d'administré le serveur à distance il faut configurer sasl. En raison de nombreux problèmes de sécurité graves la méthode d'accès GSSAPI (KERBEROS) est maintenant la valeur par défaut.

Il faut donc configuer un serveur KERBEROS dans le domaine local.

Installer les paquets suitvants.

```
dnf install -y krb5-server krb5-libs krb5-workstation
```

Configurer le fichier `/etc/krb5.conf` afin d'utiliser le domaine local:

```
# To opt out of the system crypto-policies configuration of krb5, remove the
```

```
# symlink at /etc/krb5.conf.d/crypto-policies which will not be recreated.
includedir /etc/krb5.conf.d/

[logging]
    default = FILE:/var/log/krb5libs.log
    kdc = FILE:/var/log/krb5kdc.log
    admin_server = FILE:/var/log/kadmind.log

[libdefaults]
    dns_lookup_realm = false
    ticket_lifetime = 24h
    renew_lifetime = 7d
    forwardable = true
    rdns = false
    pkinit_anchors = FILE:/etc/pki/tls/certs/ca-bundle.crt
    spake_preauth_groups = edwards25519
    default_realm = LOCALDOMAIN
    default_ccache_name = KEYRING:persistent:%{uid}

[realms]
# EXAMPLE.COM = {
#     kdc = kerberos.example.com
#     admin_server = kerberos.example.com
# }
LOCALDOMAIN = {
     kdc = 127.0.0.1
     admin_server = 127.0.0.1
}

[domain_realm]
# .example.com = EXAMPLE.COM
# example.com = EXAMPLE.COM
```

Créer la base de données à l'aide de l'utilitaire kdb5_util:

```
kdb5_util create -s
```

Démarrer Kerberos à l'aide des commandes suivantes :

```
systemctl start krb5kdc.service
systemctl start kadmin.service
systemctl enable krb5kdc.service
systemctl enable kadmin.service
```

Dans **kadmin.local** créer le principal du domaine local:

```
kadmin.local
kadmin.local: add_principal libvirt/localhost@LOCALDOMAIN
```

```
kadmin.local: quit
```

Exécuter **kinit** pour obtenir un ticket et le stocker dans un fichier de cache

```
kinit libvirt/localhost
```

Dans **kadmin.local** exporter les principal de un fichier tab:

```
kadmin.local
kadmin.local: ktadd -k /root/libvirt-localhost.tab libvirt/localhost
kadmin.local: quit
```

Placer le fichier tab dans `/etc/libvirt/krb5.tab`:

```
mv /root/libvirt-localhost.tab /etc/libvirt/krb5.tab
```

Redémarrer le serveur KDC:

```
systemctl restart krb5kdc.service
```

Enfin, vérifier que l'hyperviseur puisse être administré par tcp effectuer le test suivant:

```
 virsh  -c qemu+tcp://localhost/system nodeinfo
modèle de CPU :    x86_64
CPU :              32
Fréquence de la CPU : 2800 MHz
socket(s) CPU :    1
Coeur(s) par emplacements : 8
Thread(s) par coeur : 2
cellule(s) NUMA :   2
Taille mémoire :   131155360 KiB
```

Vagrant

L’installation et la configuration post-installation d'une machine virtuelle reste une activité relativement chronophage. Pour installer une machine virtuelle basée sur une distribution Linux minimale, il faut à chaque fois récupérer le fichier ISO de l’installateur, créer et configurer une machine virtuelle et suivre toutes les étapes du programme d’installation.

Vagrant est une surcouche en ligne de commande qui permet la mise en place d’une machine virtuelle en moins de deux minutes chrono.

En temps normal, Vagrant s’utilise plutôt avec VirtualBox, mais il est tout à fait possible de remplacer cet hyperviseur par KVM, étant donné que les performances seront bien

meilleures.

Vagrant est fourni par le dépôt tiers de Hashicorp. Créer le fichier `/etc/yum.repos.d/hashicorp.repo` comme ceci :

```
tee /etc/yum.repos.d/hashicorp.repo<<EOF
[hashicorp]
name=Hashicorp
baseurl=https://rpm.releases.hashicorp.com/RHEL/8/x86_64/stable
enabled=1
gpgcheck=1
gpgkey=https://rpm.releases.hashicorp.com/gpg
EOF
```

Malheureusement, la dernière version en date de **Vagrant** ne permet pas de construire le plugin **vagrant-libvirt**:

```
dnf --showduplicates list vagrant
Hashicorp                                       1.7 MB/s | 1.1 MB     00:00
Dernière vérification de l'expiration des métadonnées effectuée il y a
0:00:01 le mer. 07 juin 2023 08:44:46 EDT.
Paquets disponibles
vagrant.x86_64                          2.2.9-1
hashicorp
vagrant.x86_64                          2.2.10-1
hashicorp
vagrant.x86_64                          2.2.10-2
hashicorp
vagrant.x86_64                          2.2.10-3
hashicorp
vagrant.x86_64                          2.2.10-4
hashicorp
vagrant.x86_64                          2.2.11-1
hashicorp
vagrant.x86_64                          2.2.12-1
hashicorp
vagrant.x86_64                          2.2.13-1
hashicorp
vagrant.x86_64                          2.2.14-1
hashicorp
vagrant.x86_64                          2.2.15-1
hashicorp
vagrant.x86_64                          2.2.16-1
hashicorp
vagrant.x86_64                          2.2.17-1
hashicorp
vagrant.x86_64                          2.2.18-1
hashicorp
```

```
vagrant.x86_64                          2.2.19-1
hashicorp
vagrant.x86_64                          2.3.0-1
hashicorp
vagrant.x86_64                          2.3.1-1
hashicorp
vagrant.x86_64                          2.3.2-1
hashicorp
vagrant.x86_64                          2.3.3-1
hashicorp
vagrant.x86_64                          2.3.4-1
hashicorp
vagrant.x86_64                          2.3.6-1
hashicorp
```

La solution la plus viable consiste à installer la dernière version de la branche 2.2 :

```
dnf install -y vagrant-2.2.19-1 libvirt-devel
```

Il faut empêcher la mise à jour automatique du paquet vagrant dans la configuration du gestionnaire de paquets /etc/yum.conf: `exclude=vagrant`

Pour installer le plug-in pour KVM, il faut un environnement de construction disposant des outils de développement. On va donc créer un conteneur afin de disposer d'une toolchain gcc:

Dans le conteneur, les dépôts source de Rocky Linux doivent être activés :

```
cat > /etc/yum.repos.d/Rocky-Sources.repo<<'EOF'
[baseos-source]
name=BaseOS Source
baseurl=https://dl.rockylinux.org/$contentdir/$releasever/BaseOS/source/tree
/
proxy=http://proxy.infra.dgfip:3128
gpgcheck=1
enabled=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-rockyofficial

[appstream-source]
name=AppStream Source
baseurl=https://dl.rockylinux.org/$contentdir/$releasever/AppStream/source/t
ree/
```

```
proxy=http://proxy.infra.dgfip:3128
gpgcheck=1
enabled=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-rockyofficial
EOF
```

Créer un répertoire ~/vagrant-build qui sera utilisé pour la compilation des bibliothèques :

```
mkdir ~/vagrant-build
cd ~/vagrant-build/
```

Télécharger les sources des paquets krb5-libs et libssh :

```
dnf download --source krb5-libs libssh
...
(1/2): libssh-0.9.6-3.el8.src.rpm  3.3 MB/s | 1.0 MB     00:00
(2/2): krb5-1.18.2-21.el8.src.rpm  2.0 MB/s | 9.8 MB     00:04
```

Dans un premier temps, compiler et remplacer la bibliothèque **krb5** :

```
rpm2cpio krb5-*.src.rpm | cpio -idmv krb5-*.tar.gz
tar -xzf krb5-*.tar.gz
pushd krb5-*/src/
./configure
make
popd
```

Procéder de même pour **libssh** :

```
rpm2cpio libssh-*.src.rpm | cpio -imdv  libssh-*.tar.xz
tar -xJf libssh-*.tar.xz
mkdir build
pushd build/
cmake ../libssh-*  -DOPENSSL_ROOT_DIR=/opt/vagrant/embedded
make
popd
```

Sortir du conteneur et remplacer les bibliothèques installées par défaut par celles que l'on vient de compiler:

```
pushd /data/nfsshare/toolchain/gcc-el8/root/vagrant-build/krb5-*/src/
sudo cp -a lib/crypto/libk5crypto.so.3* /opt/vagrant/embedded/lib64/
popd
pushd /data/nfsshare/toolchain/gcc-el8/root/vagrant-build/build
sudo cp lib/libssh* /opt/vagrant/embedded/lib64/
popd
```

Maintenant que les bibliothèques embarquées de **Vagrant** sont à jour, installer le plug-in **0vagrant-libvirt** pour permettre à **Vagrant** d'interagir avec l'hyperviseur **KVM**:

```
vagrant plugin install vagrant-libvirt
```

Si l'installation ne trouve pas la librairie **libvirt** sur le système, il s'agit d'un problème connu et pour passer outre il suffit de passer le path de la bibliotèque à la commande ains:

```
CONFIGURE_ARGS='with-ldflags=-L/opt/vagrant/embedded/lib with-
libvirt-include=/usr/include/libvirt with-libvirt-lib=/usr/lib64'
GEM_HOME=~/.vagrant.d/gems
GEM_PATH=$GEM_HOME:/opt/vagrant/embedded/gems
PATH=/opt/vagrant/embedded/bin:$PATH vagrant plugin install
vagrant-libvirt
```

Vérifier si le plug-in s'affiche bien :

```
vagrant plugin list
vagrant-libvirt (0.10.8, global)
```

WebVirtMgr

Installation

Installer les paquets suivants:

```
dnf install -y git supervisor nginx
```

webvirtmgr est écrit avec pyhon2 il faut donc installer les modules python2-libvirt python2-libxml2 et python2-websockify non disponibles dans les dépots officiels:

```
dnf install -y python2-pip
dnf install -y /data/nfsshare/html/os/el/8/x86_64/libvirt-
python-4.5.0-1.el7.x86_64.rpm
dnf install -y /data/nfsshare/html/os/el/8/x86_64/python2-
libxml2-2.9.7-12.1.gf.el8.x86_64.rpm
dnf install -y /data/nfsshare/html/os/el/8/x86_64/python2-
websockify-0.8.0-13.el7ost.noarch.rpm
```

Télécharger la dernière release de webvirtmgr

```
git clone https://github.com/retspen/webvirtmgr.git
```

Installer les prérequis de python et paramétrer Django

```
pushd webvirtmgr/
pip2 install --proxy http://proxy.ifra.dgfip:3128 -r requirements.txt
python2 ./manage.py syncdb
python2 ./manage.py collectstatic
popd
pip2 install django --upgrade
```

La commande **syncdb** propose de créer un superuser:

```
     You just installed Django's auth system, which means you don't have any
superusers defined.
     Would you like to create one now? (yes/no): yes (Put: yes)
     Username (Leave blank to use 'admin'): admin (Put: your username or
login)
     E-mail address: username@domain.local (Put: your email)
     Password: xxxxxx (Put: your password)
     Password (again): xxxxxx (Put: confirm password)
     Superuser created successfully.
```

Paramétrage

Habituellement **WebVirtMgr** est uniquement disponible à partir de localhost sur le port 8000. Cette étape mettra WebVirtMgr à la disposition de tout le monde sur le port 80.

Déplacer le dossier webvirtmgr dans /var/www

```
cd ..
mkdir /var/www/
mv webvirtmgr /var/www/
chown -R nginx:nginx /var/www
```

Créer le fichier de configuration `/etc/nginx/conf.d/webvirtmgr.conf`:

```
cat > /etc/nginx/conf.d/webvirtmgr.conf<< 'EOF'
server {
    listen 80;

    server_name hyperviseur;

    root /var/www/webvirtmgr;

    access_log /var/log/nginx/webvirtmgr.access.log;
```

```
    error_log  /var/log/nginx/webvirtmgr.error.log;

    location / {
        try_files $uri @webvirtmgr;
    }

    location @webvirtmgr {
        proxy_pass_header Server;
        proxy_set_header Host $http_host;
        proxy_redirect off;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Scheme $scheme;
        proxy_connect_timeout 10;
        proxy_read_timeout 10;
        proxy_pass http://127.0.0.1:8000;
    }
}
EOF
```

Modifier les droits du fichier

```
chown nginx:nginx /etc/nginx/conf.d/webvirtmgr.conf
```

Invalider l'ancien bloc server:
- soit en commentant les lignes du bloc dans le fichier /etc/nginx/conf.d/default.conf
- soit directement en modifiant l'extension du fichier default.conf

Lorsque **SELinux** est activé il faut autoriser les scripts et modules HTTPD à se connecter au réseau (HTTPD Service `httpd_can_network_connect`) :

```
setsebool -P httpd_can_network_connect 1
```

Il faut également accorder au processus HTTPD l'accès aux fichiers ainsi qu'aux fichiers statiques du site avec la commande:

```
chcon -v -R --type=httpd_sys_content_t /var/www/webvirtmgr/static/
```

Relancer le service nginx et activer le démarrage automatique:

```
systemctl restart nginx
systemctl enable nginx
```

Ouvrir l'accès des ports nginx dans le Firewall

```
firewall-cmd --permanent --add-service=http
firewall-cmd --reload
```

Editer le fichier `/etc/supervisord.conf` pour y ajouter les lignes suivantes:

```
[program:webvirtmgr]
command=/usr/bin/python2 /var/www/webvirtmgr/manage.py runserver 0:8000
directory=/var/www/webvirtmgr
autostart=true
autorestart=true
stdout_logfile=/var/log/supervisor/webvirtmgr.log
redirect_stderr=true
user=nginx

[program:webvirtmgr-console]
command=/usr/bin/python2 /var/www/webvirtmgr/console/webvirtmgr-console
directory=/var/www/webvirtmgr
autostart=true
autorestart=true
stdout_logfile=/var/log/supervisor/webvirtmgr-console.log
redirect_stderr=true
user=nginx
```

Redémarrer le démon

```
systemctl restart supervisord
systemctl enable supervisord
```

Conteneurs

Podman

Développé par Redhat, cet outil de conteneur était initialement prévu comme un outil de débogage du moteur de conteneur CRI-O, spécialisé dans Kubernetes, afin de simplifier certaines tâches pour les développeurs d'applications et les administrateurs de clusters Kubernetes. Depuis lors, cependant, **Podman** est devenu un outil complet de gestion des conteneurs.

On peut facilement l'installer à partir des principales sources logicielles des distributions Linux:

```
dnf install -y podman
```

Une fois l'installation terminée, vérifier quelle version de **Podman** a été installé et si son service fonctionne sans erreur.

```
podman --version
```

On peut utiliser **Podman** sans l'exécuter en tant que service ; via **Socket**. Cependant, il offre également une intégration avec les services **Systemd** afin que les conteneurs ou les pods puissent être démarrés au démarrage du système et gérés de la même manière que d'autres services pouvant s'exécuter sur le système hôte.

Pour démarrer et activer les services Containers avec **systemd**, voici les commandes pour activer **Podman**.

```
sudo systemctl start podman
sudo systemctl enable podman
```

Vérifier l'état :

```
systemctl status podman
```

Buildah

Buildah est un outil de conteneurisation open source capable de créer des images à partir de zéro, Dockerfiles ou Containerfiles. Il suit également les spécifications de l'Open Container Initiative (OCI), ce qui rend les images Buildah à la fois polyvalentes et ouvertes.

```
sudo dnf install -y buildah slirp4netns fuse-overlayfs
```

LXD

Comme sur beaucoup de distributions (mis à part Ubuntu), LXD n'est pas disponible directement sous forme de package. On doit installer les dépôts epel (dont les paquets sont de qualité assez variable) pour obtenir snapd. Les snap sont des paquets d'installation qui viennent compléter le système de gestion de paquets standard avec des paquets plus portables d'une distribution à l'autre dans la mesure où ils sont fournis avec leurs dépendances binaires (par conséquent, ils s'appuient beaucoup moins sur la distribution sous-jacente).

```
dnf install -y snapd
systemctl enable --now snapd.socket
systemctl enable snapd
```

Il faut configurer **snap** afin qu'il puisse se connecter via un proxy:

```
snap set system proxy.https="http://proxy.infra.dgfip:3128"
snap set system proxy.http="http://proxy.infra.dgfip:3128"
```

Maintenant que snapd est installé il faut configurer le noyau pour expliciter le chargement des namespaces au démarrage du noyau (**user_namespace.enable=1**) et activer la possibilité de lancer des containers en tant que non-root (**namespace.unpriv_enable=1**). Toutes ces modifications se

font dans le fichier /etc/default/grub :

```
...

GRUB_CMDLINE_LINUX="... user_namespace.enable=1 namespace.unpriv_enable=1
..."

...
```

Les changements apportés à `/etc/default/grub` requièrent de régénérer le fichier `grub.cfg` comme suit :

Sur les machines basées BIOS, exécuter la commande suivante en tant qu'utilisateur root :

`grub2-mkconfig -o /boot/grub2/grub.cfg`

Sur les machines basées UEFI, exécuter la commande suivante en tant qu'utilisateur root :

`grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg`

```
grub2-mkconfig -o /boot/efi/EFI/rocky/grub.cfg
Generating grub configuration file ...
Adding boot menu entry for EFI firmware configuration
done
```

Enfin, il faut définir un nombre maximum de namespaces par utilisateur conséquent:

```
echo "user.max_user_namespaces=3883" > /etc/sysctl.d/99-userns.conf
```

Redémarrer la machine et procéder à l'installation de LXD :

```
snap install lxd

2019-03-07T22:43:22+01:00 INFO Waiting for restart...

lxd 3.10 from Canonical✓ installed
```

La configuration des containers passe par une commande d'initialisation qui permet de créer un profil par défaut. Sauf que nous allons tout faire nous même, donc nous allons répondre non partout.

```
lxd init
```

```
Would you like to use LXD clustering? (yes/no) [default=no]:
Do you want to configure a new storage pool? (yes/no) [default=yes]:
Name of the new storage pool [default=default]: lxd_zpool
Name of the storage backend to use (btrfs, ceph, dir, lvm, zfs)
[default=zfs]:
Create a new ZFS pool? (yes/no) [default=yes]: no
Name of the existing ZFS pool or dataset: zpool_lxd
Would you like to connect to a MAAS server? (yes/no) [default=no]:
Would you like to create a new local network bridge? (yes/no) [default=yes]:
What should the new bridge be called? [default=lxdbr0]:
What IPv4 address should be used? (CIDR subnet notation, “auto” or “none”)
[default=auto]:
What IPv6 address should be used? (CIDR subnet notation, “auto” or “none”)
[default=auto]: none
Would you like the LXD server to be available over the network? (yes/no)
[default=no]: yes
Address to bind LXD to (not including port) [default=all]:
Port to bind LXD to [default=8443]:
Would you like stale cached images to be updated automatically? (yes/no)
[default=yes]:
```

Le fichier YAML “lxd init” ressemble à ceci:

```
config:
  core.https_address: '[::]:8443'
networks:
- config:
    ipv4.address: auto
    ipv6.address: none
  description: ""
  name: lxdbr0
  type: ""
  project: default
storage_pools:
- config:
    source: zpool_lxd
  description: ""
  name: lxd_zpool
  driver: zfs
profiles:
- config: {}
  description: ""
  devices:
    eth0:
      name: eth0
      network: lxdbr0
      type: nic
    root:
      path: /
      pool: lxd_zpool
```

```
      type: disk
  name: default
projects: []
cluster: null
```

OpenvSwitch

Installation de openvswitch

```
dnf --enablerepo=devel install openvswitch2.13
```

Démarrer le service openvswitch

Après l'installation, démarrer manuellement le service openvswitch.

```
systemctl enable openvswitch --now
Created symlink /etc/systemd/system/multi-
user.target.wants/openvswitch.service →
/usr/lib/systemd/system/openvswitch.service.
```

Vérifier l'état du service openvswitch

```
----

-bash: $ : commande introuvable
[root@localhost ~]# systemctl status openvswitch
● openvswitch.service - Open vSwitch
   Loaded: loaded (/usr/lib/systemd/system/openvswitch.service; enabled;
vendor>
   Active: active (exited) since Fri 2023-06-02 07:21:59 EDT; 58s ago
  Process: 15879 ExecStart=/bin/true (code=exited, status=0/SUCCESS)
 Main PID: 15879 (code=exited, status=0/SUCCESS)

juin 02 07:21:59 localhost.localdomain systemd[1]: Starting Open vSwitch...
juin 02 07:21:59 localhost.localdomain systemd[1]: Started Open vSwitch.
lines 1-8/8 (END)
```

Créer et configurer un pont OVS

Avant de créer un pont, il faut activer le routage IP en définissant les paramètres du noyau lors de l'exécution à l'aide de sysctl.

```

```

```
sudo tee /etc/sysctl.d/iprouting.conf<<EOF
net.ipv4.ip_forward=1
net.ipv6.conf.all.forwarding=1
EOF
```

Appliquer les paramètres :

```
sudo sysctl --system
```

Confirmer les nouveaux paramètres :

```
sysctl net.ipv4.ip_forward

net.ipv4.ip_forward = 1
```

```
sysctl net.ipv6.conf.all.forwarding

net.ipv6.conf.all.forwarding = 1
```

La prochaine étape est la création d'un pont OVS, nommé **br-ext** avec **NetworkManager**:

Le plugin **ovs** doit être installé et activé afin de permettre à **NetworkManager** de piloter **OpenVswitch**:

```
dnf install NetworkManager-ovs
systemctl restart NetworkManager
```

Créer un pont avec une seule interface interne, l'ordre de l'exécution des commandes est important, afin de grantir le bon respect il faut mettre les commandes dans un fichier exécutable:

```
cat > ovs-brext <<'EOF'
#!/bin/bash
nmcli conn add type ovs-bridge conn.interface br-ext con-name br-ext
nmcli conn add type ovs-port conn.interface br-ext master br-ext con-name
ovs-port-br-ext
nmcli conn add type ovs-interface slave-type ovs-port con.interface br-ext
master ovs-port-br-ext con-name ovs-if-br-ex ipv4.method manual ipv4.address
10.13.254.153 ipv4.gateway 10.13.254.7 ipv4.dns 10.154.59.104,10.156.32.33
mv /etc/sysconfig/network-scripts/ifcfg-ens1f0 /etc/sysconfig/network-
scripts/ifbkp-ens1f0
nmcli conn add type ovs-port conn.interface ens1f0 master br-ext con-name
ovs-port-ens1f0
nmcli conn add type ethernet conn.interface ens1f0 master ovs-port-ens1f0
con-name ovs-if-ens1f0
EOF
```

Exécuter le script:

```
bash ./ovs-brext
```

Redémarrer le réseau

```
nmcli conn reload
```

Vérifier l'état des interfaces avec `nmcli conn`:

```
NAME             UUID                                  TYPE           DEVICE
ovs-if-br-ex     a78f8a2e-a1fd-4998-9d1b-5ee75b046017  ovs-interface  br-ext
virbr0           34e97c7b-7089-4cac-8fe3-9dd1daa5a42b  bridge         virbr0
br-ext           125c752e-f8db-46e7-9f95-d5c9d32df073  ovs-bridge     br-ext
ovs-if-ens1f0    deff16c3-7575-459e-9cd2-8c608ccb79b9  ethernet       ens1f0
ovs-port-br-ext  85273f96-1802-4900-a5de-25175cdbf0f4  ovs-port       br-ext
ovs-port-ens1f0  cf424ec4-d56b-415e-9863-6773518e8dfa  ovs-port       ens1f0
```

Voici les informations d'adresse IP telles qu'on les a configuré:

```
ip ad show dev br-ex

18: br-ex: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/ether d4:f5:ef:7b:2c:18 brd ff:ff:ff:ff:ff:ff
    inet 10.13.254.151/24 brd 10.13.254.255 scope global noprefixroute br-ex
       valid_lft forever preferred_lft forever
    inet6 fe80::ca85:e810:70ac:9285/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Enfin, créer l'interface interne sur lequel on va connecter les VM:

```
cat > ovs-brint <<'EOF'
#!/bin/bash
nmcli conn add type ovs-bridge conn.interface br-int con-name br-int
nmcli conn add type ovs-port conn.interface br-int master br-int con-name
ovs-port-br-int
nmcli conn add type ovs-interface slave-type ovs-port conn.interface br-int
master ovs-port-br-int con-name ovs-if-br-int ipv4.method disabled
ipv6.method disabled
EOF
```

Exécuter le script:

```
```

```
bash ./ovs-brint
```

Vérifier l'état des interfaces avec `nmcli conn`:

```
NAME                 UUID                                  TYPE
DEVIC>
ovs-if-br-ex         788160b6-5568-4b40-9247-d5d7002874bc  ovs-interface
br-ex>
virbr0               add11217-5557-4692-9639-9147b1e5c9de  bridge
virbr>
br-ext               047ee4ee-1db0-4532-8eb9-a6760ea9e972  ovs-bridge
br-ex>
br-int               b0922124-5b25-408b-bdaa-f53206ddbe5f  ovs-bridge
br-in>
ovs-if-ens1f0        c48f16bb-910a-4d06-9956-2ef8fc9fd8f6  ethernet
ens1f>
ovs-if-patch-br-ext  c144506c-1fc6-4099-820a-d7a500beec4c  ovs-interface
patch>
ovs-if-patch-br-int  7457fc66-0f7e-4717-b964-b33f78949c02  ovs-interface
patch>
ovs-port-br-ext      5637d602-93fb-4f25-91eb-c49adab890a0  ovs-port
br-ex>
ovs-port-br-int      99eea92f-a2d7-40c4-83ae-41eddf9c57a8  ovs-port
br-in>
ovs-port-ens1f0      f557ae24-5c0a-4e74-afd2-dfe0940942b5  ovs-port
ens1f>
patch-br-ext         c2e20367-ff37-4c7b-8e05-8c2619cf36db  ovs-port
patch>
patch-br-int         9a9e017d-a383-42b8-b07e-534902087f5b  ovs-port
patch>
```

```
 ovs-vsctl show
7073e007-88a6-4e48-9eb8-a3cdbec06f40
    Bridge br-ext
        Port br-ext
            Interface br-ext
                type: internal
        Port ens1f0
            Interface ens1f0
                type: system
    ovs_version: "2.17.2"
```

Créer le lien interne permettant de donner l'accès des interfaces internes vers l'extérieur:

```
cat > ovs-patch <<'EOF'
#!/bin/bash
nmcli conn add type ovs-port conn.interface patch-br-int master br-int con-
name patch-br-int
```

```
nmcli conn add type ovs-interface slave-type ovs-port conn.interface patch-br-int master patch-br-int con-name ovs-if-patch-br-int ovs-interface.type patch ovs-patch.peer patch-br-ext
nmcli conn add type ovs-port conn.interface patch-br-ext master br-ext con-name patch-br-ext
nmcli conn add type ovs-interface slave-type ovs-port conn.interface patch-br-ext master patch-br-ext con-name ovs-if-patch-br-ext ovs-interface.type patch ovs-patch.peer patch-br-int
EOF
```

Exécuter le script:

```
bash ./ovs-patch
```

Créer un pont logiciel non attaché ou lié à une interface hôte spécifique pour **qemu**:

```
nmcli connection add type bridge ifname br-tun con-name br-tun  ipv4.method 'shared' ipv4.addresses "10.13.254.154/30"
```

La directive **ipv4.method shared** signifie que le NAT va être activé pour les invités connectés au bridge.

Créer un scripte **qemu-ifup** (qui ne fait rien mais il est requis lors du processus de build qemu):

```
cat > /etc/qemu-ifup <<'EOF'
#!/bin/bash`
EOF
```

Cockpit

Cockpit a été créé par RedHat dans le but principal de faciliter l'administration du serveur. Il s'agit d'un outil Web basé sur une interface graphique qui aide à effectuer diverses tâches d'administration.

Avec **Cockpit**, les mêmes API système sont utilisées que celles du terminal, et les tâches effectuées sur le terminal sont rapidement répercutées sur la console Web.

Installation de la console Web Cockpit

Par défaut, **Cockpi**t n'est pas installé sur votre système Rocky Linux 8. Il faut donc l'installer sur Rocky Linux 8 en utilisant la commande :

```
```

```
dnf install cockpit
```

Avec **Cockpit** installé, il faut démarrer et activer le service **cockpit.socket** pour pouvoir connecter le système via la console Web.

```
sudo systemctl start cockpit.socket
sudo systemctl enable --now cockpit.socket
```

Vérifier l'état du service :

```
systemctl status cockpit.socket
● cockpit.socket - Cockpit Web Service Socket
   Loaded: loaded (/usr/lib/systemd/system/cockpit.socket; enabled; vendor
pres>
   Active: active (running) since Mon 2023-07-10 08:41:55 EDT; 51min ago
     Docs: man:cockpit-ws(8)
   Listen: [::]:9090 (Stream)
    Tasks: 0 (limit: 819472)
   Memory: 688.0K
   CGroup: /system.slice/cockpit.socket

juil. 10 08:41:55 localhost.localdomain systemd[1]: Starting Cockpit Web
Servic>
juil. 10 08:41:55 localhost.localdomain systemd[1]: Listening on Cockpit Web
Se>
lines 1-11/11 (END)
```

Ouvrir l'accès des ports 9090 dans le Firewall

```
firewall-cmd --permanent --add-port=9090/tcp
firewall-cmd --reload
```

Afin de permettre à cockpit de gérer les mises à jour des paquets, il faut rajouter dans les fichiers des dépôts actifs dans `/etc/yum.repos.d` le proxy:

```
cd /etc/yum.repos.d
sed -i
's/enabled=1/enabled=1\nproxy="http:\/\/proxy.infra.dgfip:3128"/g'
*.repo
```

Installation des modules

- **cockpit-podman**: pour permettre à **Cockpit** d'extraire ou de télécharger des images **Podman** et de gérer les conteneurs **Podman**.

- **cockpit-machines**: pour gérer les machines virtuelles.

```
dnf install -y cockpit-podman cockpit-machines
```

Bien que **Podman** ne soit pas un démon, un service Systemd est disponible pour fournir un accès à l'API **Podman** afin que **Cockpit** puisse interagir directement avec **Podman**.