Nouvel Hyperviseur v9.02

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

Ivalider les dépôts par défaut:

```
```

```
sed -i 's/enabled=1/enabled=0/g' /etc/yum.repos.d/*.repo
```

Définir les dépôts du socle:

```
cat > /etc/yum.repos.d/rocky-socle.repo<<"EOF"

# rocky-socle.repo

[addons-socle]
name=socle-2024-9.02-x86_64 : addons-socle-2024-9.02-x86_64-20230522
baseurl=http://repo.infra.dgfip/pub/socles/socle-2024-9.02-x86_64/repos/addo
ns-socle-2024-9.02-x86_64-20230522
gpgcheck=0
enabled=1

[appstream-socle]
name=socle-2024-9.02-x86_64 : appstream-socle-2024-9.02-x86_64-20230522
baseurl=http://repo.infra.dgfip/pub/socles/socle-2024-9.02-x86_64/repos/apps
tream-socle-2024-9.02-x86_64-20230522
gpgcheck=0
enabled=1

[baseos-socle]
name=socle-2024-9.02-x86_64 : baseos-socle-2024-9.02-x86_64-20230522
baseurl=http://repo.infra.dgfip/pub/socles/socle-2024-9.02-x86_64/repos/base
os-socle-2024-9.02-x86_64-20230522
gpgcheck=0
enabled=1

[crb-socle]
name=socle-2024-9.02-x86_64 : crb-socle-2024-9.02-x86_64-20230522
baseurl=http://repo.infra.dgfip/pub/socles/socle-2024-9.02-x86_64/repos/crb-
socle-2024-9.02-x86_64-20230522
gpgcheck=0
enabled=1

[elrepo-socle]
name=socle-2024-9.02-x86_64 : elrepo-socle-2024-9.02-x86_64-20230522
baseurl=http://repo.infra.dgfip/pub/socles/socle-2024-9.02-x86_64/repos/elre
po-socle-2024-9.02-x86_64-20230522
gpgcheck=0
enabled=1

[epel-socle]
name=socle-2024-9.02-x86_64 : epel-socle-2024-9.02-x86_64-20230522
baseurl=http://repo.infra.dgfip/pub/socles/socle-2024-9.02-x86_64/repos/epel
-socle-2024-9.02-x86_64-20230522
gpgcheck=0
enabled=1
```

```
[extras-socle]
name=socle-2024-9.02-x86_64 : extras-socle-2024-9.02-x86_64-20230522
baseurl=http://repo.infra.dgfip/pub/socles/socle-2024-9.02-x86_64/repos/extras-socle-2024-9.02-x86_64-20230522
gpgcheck=0
enabled=1

[ha-socle]
name=socle-2024-9.02-x86_64 : ha-socle-2024-9.02-x86_64-20230522
baseurl=http://repo.infra.dgfip/pub/socles/socle-2024-9.02-x86_64/repos/ha-socle-2024-9.02-x86_64-20230522
gpgcheck=0
enabled=1

[plus-socle]
name=socle-2024-9.02-x86_64 : plus-socle-2024-9.02-x86_64-20230522
baseurl=http://repo.infra.dgfip/pub/socles/socle-2024-9.02-x86_64/repos/plus-socle-2024-9.02-x86_64-20230522
gpgcheck=0
enabled=1

[addons-socle]
name=socle-2024-9.02-x86_64 : rs-socle-2024-9.02-x86_64-20230522
baseurl=http://repo.infra.dgfip/pub/socles/socle-2024-9.02-x86_64/repos/rs-socle-2024-9.02-x86_64-20230522
gpgcheck=0
enabled=1

[tina-socle]
name=socle-2024-9.02-x86_64 : tina
baseurl=http://repo.infra.dgfip/pub/socles/socle-2024-9.02-x86_64/repos/tina
gpgcheck=0
enabled=0

[tools-socle]
name=socle-2024-9.02-x86_64 : tools-socle-2024-9.02-x86_64
baseurl=http://repo.infra.dgfip/pub/socles/socle-2024-9.02-x86_64/repos/tools-socle-2024-9.02-x86_64
gpgcheck=0
enabled=1
EOF
```

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 libvirt-devel
```

Politique SElinux

De nombreux problèmes signalés sont liés à l'activation de SElinux. Jusqu'à présent, la solution consistait à désactiver complètement SElinux, mais il s'agissait plus d'une solution de contournement qu'autre chose. Pour permettre les contrôles ICMP par HTTPD sur le serveur, une politique SElinux doit être utilisée.

Fondamentalement, cela permet l'ouverture de sockets IP bruts pour les utilisateurs non root, nécessaires à l'exécution de la commande ping.

- Créer le fichier `http_ping.tt`:

```
cat > http_ping.tt <<'EOF'
module http_ping 1.0;

require {
type httpd_t;
class capability net_raw;
class rawip_socket { getopt create setopt write read };
}

#============= httpd_t ==============
allow httpd_t self:capability net_raw;
allow httpd_t self:rawip_socket { getopt create setopt write read };
EOF
```

- Exécuter les commandes suivantes (en tant qu'utilisateur root) :

```
checkmodule -M -m -o http_ping.mod http_ping.tt
semodule_package -o http_ping.pp -m http_ping.mod
semodule -i http_ping.pp
```

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 100% vg1
```

Créer les systèmes de fichiers:

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

Créer le dossier de montage:

```
mkdir -pv /data
```

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

```
/dev/vg1/vol1           /data                   xfs     defaults        0 0
```

Rechargement du démon et montage manuel des systèmes de fichiers:

```
systemctl daemon-reload
mount /data
```

Vérifier le montage correcte des systèmes de fichiers:

```
df -h
/dev/mapper/vg1-vol1   1,1T    7,7G  1,1T   1% /data
```

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
```

Créer l'espace de stockage des 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/
```

Activer le mode d'écoute 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`

Historiquement, libvirtd était responsable de la création des sockets sur lesquelles il écoutait. Par défaut, il créerait des sockets UNIX pour un accès local, et si –listen est donné, il créerait des sockets TCP ou TLS en fonction des paramètres `listen_tls` et `listen_tcp` dans le fichier `/etc/libvirt/libvirtd.conf`;
Désormais lee démon **libvirtd** préfère utiliser l'activation du socket **systemd**. Cela signifie que **systemd** crée les sockets UNIX et transmet les FD pré-ouverts à **libvirtd** lors de son démarrage. Le paramètre `--listen` n'est **PAS** honoré lorsque l'activation de socket est utilisée. L'utilisation de `--listen` empêchera le démarrage de **libvirtd**. Pour revenir au mode traditionnel, il faut masquer tous les fichiers socket:

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

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
```

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**.

```
systemctl start podman
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.

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

LXC

À ne pas confondre avec l'outil client de ligne de commande lxc fourni par LXD, LXC (Linux Container) est une technologie de virtualisation au niveau du système d'exploitation qui utilise une API puissante et d'autres outils pour permettre aux utilisateurs de créer et de gérer de manière transparente des conteneurs et des machines virtuelles dans un seul hôte. Il comprend des modèles, un langage d'outils et des liaisons de bibliothèque.

```
dnf install -y lxc lxc-templates lxc-extra
```

LXC utilise une interface de pont pour fournir des capacités de mise en réseau aux conteneurs. Configurer le pont en modifiant le fichier de configuration.

Ouvrir le fichier de configuration `/etc/lxc/default.conf` dans un éditeur de texte pour configurer le réseau LXC:

```
lxc.net.0.type = veth
lxc.net.0.link = br-tun
lxc.net.0.flags = up
lxc.net.0.hwaddr = 00:16:3e:xx:xx:xx
```

Enregistrer les modifications et quitter l'éditeur de texte.

Démarrer et activer les services LXC

Pour démarrer les services LXC et s'assurer qu'ils démarrent automatiquement au démarrage, exécuter les commandes suivantes :

```
systemctl start lxc.service
systemctl enable lxc.service
```

OpenvSwitch

Installation de openvswitch

```
dnf install -y openvswitch2.17 NetworkManager-ovs
```

Sur une plate-forme CentOS compatible, openvswitch est fournit par le SIG CentOS NFV (Network Function Virtualization), il faut donc installer **centos-release-nfv-openvswitch**:
`dnf install centos-release-nfv-openvswitch`

Démarrer le service openvswitch:

Après l'installation, démarrer manuellement le service openvswitch.

```
systemctl enable openvswitch --now
```

Vérifier l'état du service openvswitch

```


----

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)
```

Redémarrer le service NetworkManager:

```
systemctl restart NetworkManager
```

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.

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

Appliquer les paramètres :

```
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**:

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.

```
systemctl start cockpit.socket
```

```
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
```

Cockpit ne permet pas à l'utilisateur root de se connecter à la console d'administration (cf. `/etc/cockpit/disallowed-users`). Il faut donc créer au moins un utilisateur avec les droits d'administration (sudo):

```
adduser admin
passwd admin
usermod -aG wheel admin
```

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**.