Installation hyperviseur KVM sur Rocky8

Table of Contents

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
```

Configuration de libvirt

Le démon libvirtd est capable de démarrer en deux modes.

1. En mode traditionnel, il créera et écoutera lui-même sur les sockets UNIX. Si le paramètre

–listen est donné, il écoutera également sur les sockets TCP/IP, selon les options listen*tcp et listen*tls dans /etc/libvirt/libvirtd.conf
2. En mode d'activation de socket, il s'appuiera sur systemd pour créer et écouter sur UNIX, et éventuellement TCP/IP, des sockets et les transmettre en tant que descripteurs de fichiers pré-ouverts. Dans ce mode, il n'est pas permis de passer le paramètre –listen, et la plupart des options de configuration liées aux sockets dans /etc/libvirt/libvirtd.conf n'auront plus aucun effet.

Le mode d'activation du socket est généralement le mode par défaut lors de l'exécution sur un système d'exploitation hôte qui utilise systemd. Pour revenir au mode traditionnel, tous les fichiers socket unit doivent être masqués :

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

Ce mode d'activation requiert un ensemble de certificats auxquels attaché les rerveur. Dans un premier temps 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 ACME Organization, 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 = Name of your organization
cn = compute1.libvirt.org
dns_name = compute1
dns_name = compute1.libvirt.org
ip_address = 10.0.0.74
ip_address = 192.168.1.24
```

```
ip_address = 2001:cafe::74
ip_address = fe20::24
tls_www_server
encryption_key
signing_key
EOF
```

Le champ **cn** doit faire référence au nom d'hôte public complet du serveur. Pour les données d'extension SAN, il doit également y avoir un ou plusieurs champs **dns_name** contenant tous les noms d'hôtes possibles pouvant être raisonnablement utilisés par les clients pour atteindre le serveur, avec et sans qualificatifs de nom de domaine. Si les clients sont susceptibles de se connecter au serveur par adresse IP, alors un ou plusieurs champs **ip_address** doivent également être ajoutés.

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
```

Pour activer le mode traditionnel, il faut modifier le fichier `/etc/sysconfig/libvirtd` pour décommenter la ligne:

```
listen_tcp = 1
```

Auparavant, **DIGEST-MD5** était défini comme mécanisme par défaut pour libvirt. Mais en raison de nombreux problèmes de sécurité graves cela ne doit plus être utilisé. Ainsi **GSSAPI** (KERBEROS) est maintenant la valeur par défaut. Pour rétablir le mécanisme **DIGEST-MD5** il faut:

- installer le plugin digest-md5: `yum install cyrus-sasl-md5`

- Remplacer le mécanisme **gssapi** par **digest-md5** dans le fichier /etc/sasl2/libvirt.conf: `mech_list: digest-md5`

- décommenter dans ce fichier la ligne `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
```

Afin de vérifierr que l'hyperviseur puisse être administré à distance effectuer le test suivant:

```
     virsh -c qemu+tcp://IP_address/system nodeinfo
     ------------------------------------------------------------------------
-------
     Please enter your authentication name: fred
     Please enter your password: xxxxxx
     CPU model: x86_64
     CPU(s): 2
     CPU frequency: 2611 MHz
     CPU socket(s): 1
     Core(s) per socket: 2
     Thread(s) per core: 1
     NUMA cell(s): 1
     Memory size: 2019260 kB
```

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
```

Le service **libvirt-guests** gère le redémarrage automatique des machines virtuelles en cas de redémarrage de l'hyperviseur.

Pour activer la journalisation de **libvirtd** ajouter dans le fichier /etc/libvirt/libvirtd.conf:

```
log_level = 1
log_outputs="1:file:/var/log/libvirt/libvirtd.log"
```

'level' est le niveau minimal auquel les messages correspondants doivent être connectés:
- 1: DEBUG
- 2: INFO

- 3: WARNING
- 4: ERROR

Configuration d’un interface réseau

Dans la configuration par défaut, les machines virtuelles sont associées au réseau virtuel 192.168.122.0/24. Elles peuvent communiquer entre elles et accéder au monde extérieur. En revanche, la configuration NAT (Network Address Translation) ne permet pas d’accéder à ces machines depuis l’extérieur. Une solution consiste ici à mettre en place un pont réseau (bridge) qui permet d’intégrer la ou les machines virtuelles au réseau de l’hyperviseur.

Afficher la configuration réseau et notez les paramètres :

```
ip --brief addr show
```

Lancer **NetworkManager TUI**:

```
LANG=fr_FR.UTF-8 && nmtui
```

Configurer le bridge **br0**:

- Sélectionner **Edit a connection**.
- Add > Bridge > Create
- **Profile name : BRIDGE**
- **Device : br0**
- Bridge Slaves > Add
- Slave connection > Ethernet > Create
  - **Profile name : LAN**
  - **Device : enp1s0**
  - Confirmer par **OK**.
- Décocher Enable STP (Spanning Tree Protocol).
- Passer **IPv6 CONFIGURATION** de **Automatic** à **Ignore**.
- Confirmer par **OK**.
- Revenir à la fenêtre principale.
- Sélectionner **Activate a connection**.
- Les connexions doivent être actives (*).
- Revenir à la fenêtre principale.
- Quitter NetworkManager TUI.

Vérifier la configuration du bridge :

```
ip --brief addr show
```

Si l’interface br0 dispose bien de sa propre adresse IP, on peut retourner dans

NetworkManagerTUI et supprimer toutes les autres connexions comme Wired connection 1, System eth0 ou virbr0.

Pour installer des machines virtuelles et interagir avec elles, on a le choix. Mais on peut se simplifier la vie en utilisant Virtual Machine Manager (virt-manager), une interface graphique relativement simple à manipuler. Le paquet correspondant est également fourni par les dépôts officiels de la distribution.

Si vous faites tourner KVM sur une station de travail graphique, ajoutez votre utilisateur au groupe libvirt pour éviter les demandes d'authentification à répétition :

```
usermod -aG libvirt kikinovak
```

Lorsque l'hyperviseur est installé sur un serveur distant et que l'on souhaite piloter à distance, on peut installer **Virtual Machine Manager** sans toute la partie hyperviseur comme ceci :

```
dnf install --setopt=install_weak_deps=false virt-manager
```

WebVirtMgr

Installation de nginx

Certains prérequis ne sont disponibles que sur les dépôts EPEL. Vérifier la présence de ceux-ci par yum update

Si EPEL n'apparaît pas activer les dépôts d'EPEL

```
dnf install -y git python3-pip python3-libvirt python3-libxml2 python3-
websockify supervisor nginx
```

Les modules python libvirt libxml2 et websockify sont nécessaires mais webvirtmgr est écrit avec pyhon2 il faut donc installer les modules compatibles:

```
dnf install python2-pip
yum install
https://rpmfind.net/linux/centos/7.9.2009/os/x86_64/Packages/libvirt-python-
4.5.0-1.el7.x86_64.rpm
yum install
http://mirror.ghettoforge.org/distributions/gf/el/8/plus/x86_64/python2-libx
ml2-2.9.7-12.1.gf.el8.x86_64.rpm
yum install
```

```
https://update.cs2c.com.cn/NS/V7/V7Update6/os/adv/lic/appstore/openstack/aarch64/Packages/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

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

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.
```

Pour créer un superuser admin utiliser la commande suivante:

```
python2 ./manage.py createsuperuser
```

Paramétrage de Nginx

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.

Cela signifie que tout le monde entre l'utilisateur et le serveur (personnes sur le même wifi, routeur local, fournisseur d'accès, le fournisseur de serveurs, backbones etc.) peut voir vos informations de connexion en texte clair!.

Au lieu de cela, on peut également sauter cette étape complètement et rediriger simplement le port 8000 sur la machine locale via SSH. Cela est beaucoup plus sûr parce WebVirtMgr ne sont pas disponibles au public plus et que l'on ne peut y accéder via une connexion cryptée.:

```
ssh user@server:port -L localhost:8000:localhost:8000 -L
localhost:6080:localhost:6080
```

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
```

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
/chemin/fichiers/statiques/du/site/
```

Relancer le service nginx

```
service nginx restart
```

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

```
service supervisord restart
```

OpenvSwitch

Installation de openvswitch

```
dnf --enablerepo=devel install openvswitch2.17
```

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

```
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)
```

L'utilitaire **ovs-vsctl** est fourni pour interroger et configurer **ovs-vswitchd**. Il fournit une interface de haut niveau pour la configuration de la base de données de configuration Open vSwitch.

Pour vérifier la version d'OVS, exécuter la commande suivante:

```
ovs-vsctl show
```

Créer et configurer un pont OVS

Dans une configuration réseau typique utilisant OVS, un pont créé aura une connexion directe à une interface réseau dédiée dans le système hôte. Cela limite le pont et les invités connectés à utiliser uniquement cette interface hôte.

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

```
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-ext ipv4.method manual ipv4.addresses
xx.xx.xxx.xxx/24 \
      ipv4.gateway xx.xx.xxx.x ipv4.dns xx.xxx.xx.xxx,xx.xx.xx.xx
```

Avant d'ajouter l'interface, les périphériques Bridge et Port semblent actifs, mais ne sont pas encore configurés dans OVSDB, il faut créer un port même pour une seule interface. On peut inspecter les résultats avec ovs-vsctl show.

Pour ajouter un port physique au pont br-ext, le fichier de configuration /etc/sysconfig/network-scripts/ifcfg-xxxx doit être supprimé avant le rechargement de nmcli

Ajouter une interface Linux à un Bridge

```
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
```

Redémarrer le réseau

```
nmcli conn reload
```

Les connexions doivent resembler à celà:

```
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 avons configurées.

```
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 xx.xx.xxx.xxx/24 brd xx.xx.xxx.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:

```
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
```

L'ordre de création des connections est important, afin de faciliter le paramétrage on peut utiliser des scriptes bash:

```
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 xx.xx.xxx.xxx ipv4.gateway
xx.xx.xxx.x ipv4.dns xx.xxx.xx.xxx,xx.xxx.xx.xx
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
```

```
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
```

```
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
```

```
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"
```

Voici les informations d'adresse IP telles qu'on les avons configurées.

```
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 xx.xx.xxx.xxx/24 brd xx.xx.xxx.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
```

La pile TCP/IP du système hôte peut gérer le routage du trafic sortant vers l'interface appropriée en fonction de l'adresse IP ou du sous-réseau de destination. Pour cela il faut créer un pont logiciel non attaché ou lié à une interface hôte spécifique:

```
nmcli connection add type bridge ifname br-tun con-name br-tun
ipv4.method manual ipv4.addresses "xx.xx.xxx.xxx/29"
```

et un scripte **qemu-ifup** (qui ne fait rien mais il est requis par qemu):

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

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

```
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
```