Installs and Stuff

Application installs for Rocky Linux 8.8 (and most likely variants). Also some reference material.

- Install: Open vSwitch (OVS)
- Install: KVM / QEMU
- Reference: KVM Networks

Install: Open vSwitch (OVS)

Here are the steps to get Open vSwitch loaded up from source (don't let that scare you, we are still going to use RPMs).
There are other directions out there. Now, this will be one of them.

NOTE: I have included some cleanup actions and some manual dependency installs. Just use what you need if not all of it.

```
# REMOVE OVS IF INSTALL WITH DISTRO PACKAGES
systemctl stop openvswitch && systemctl disable openvswitch
systemctl stop ovs-vswitchd && systemctl disable ovs-vswitchd
systemctl stop ovsdb-server && systemctl disable ovsdb-server

dnf remove openvswitch* *openvswitch -y

rm -f /etc/sysconfig/openvswitch
rm -fR /etc/openvswitch*
rm -fR /var/run/openvswitch*
rm -fR /var/log/openvswitch*
```

```
# CLONE OVS SOURCE FROM GITHUB
cd /opt
git clone https://github.com/openvswitch/ovs.git

# ENABLE IPROUTING
cat > /etc/sysctl.d/iprouting.conf << "EOF"
net.ipv4.ip_forward=1
net.ipv4.conf.all.rp_filter = 2
EOF
```

```
# CLEANED UP OLD PACKAGES, DNF CACHE, AND INSTALL DEPS
dnf autoremove
dnf clean all
dnf makecache

# REMOVED OLD KERNELS
dnf remove --oldinstallonly --setopt installonly_limit=2 kernel -y
```

```
# REINSTALLED LATEST KERNEL AND MODULES
dnf reinstall kernel kernel-core kernel-modules -y

# INSTALLED DEV TOOLS AND BUILD PACKAGES
dnf install @'Development Tools' dnf-plugins-core rpm-build make automake -y

# INSTALLED DEPS
dnf install python3-six python3-sphinx libunwind-devel unbound-devel \
libpfm-devel python3-libpfm libcap-ng libcap-ng-devel libpcap-devel \
bpf* libbpf* libnuma* numa*
```

```
# REBOOTED
reboot
```

NOTE: If you want your RPM's to be compiled with DPDK, you will need to also go grab the DPDK version 22.x source and build that as well. Rocky 8.8 only has the 21.x packages available. I opted to not compile it in.

Open vSwitch with DPDK

```
# OVS CONFIGURE, DEP CHECK AND RPM BUILD
cd /opt/ovs

./boot.sh
./configure --prefix=/usr --localstatedir=/var --sysconfdir=/etc

sed -e 's/@VERSION@/3.2.90/' rhel/openvswitch-fedora.spec.in > /tmp/ovs.spec
dnf builddep /tmp/ovs.spec
rm -f /tmp/ovs.spec

make rpm-fedora RPMBUILD_OPT="--with check"

ls -al rpm/rpmbuild/RPMS/x86_64/
```

```
# INSTALL OVS
cd rpm/rpmbuild/RPMS/x86_64/
```

```
dnf install \
openvswitch-3.2.90-1.el8.x86_64.rpm \
network-scripts-openvswitch-3.2.90-1.el8.x86_64.rpm \
openvswitch-devel-3.2.90-1.el8.x86_64.rpm
```

Install: KVM / QEMU

Here are the steps to get KVM/QEMU installed.

First check and make sure that virtualization is enabled in the bios and seen by the system

```
lscpu | grep Virtualization
```

Should look something like this

```
[root@bs ~]# lscpu | grep Virtualization
Virtualization:      VT-x
Virtualization type: full
```

Can also use this command and look for 'vmx' or 'svm' in the output.

```
cat /proc/cpuinfo | egrep "vmx|svm"
```

Should look something like this

```
tsc_known_freq pni pclmulqdq vmx ssse3 cx16 pcid sse4_1 sse4_2 x2apic
tsc_known_freq pni pclmulqdq vmx ssse3 cx16 pcid sse4_1 sse4_2 x2apic
tsc_known_freq pni pclmulqdq vmx ssse3 cx16 pcid sse4_1 sse4_2 x2apic
tsc_known_freq pni pclmulqdq vmx ssse3 cx16 pcid sse4_1 sse4_2 x2apic
tsc_known_freq pni pclmulqdq vmx ssse3 cx16 pcid sse4_1 sse4_2 x2apic
tsc_known_freq pni pclmulqdq vmx ssse3 cx16 pcid sse4_1 sse4_2 x2apic
tsc_known_freq pni pclmulqdq vmx ssse3 cx16 pcid sse4_1 sse4_2 x2apic
tsc_known_freq pni pclmulqdq vmx ssse3 cx16 pcid sse4_1 sse4_2 x2apic
```

Once that is validated, load up some KVM

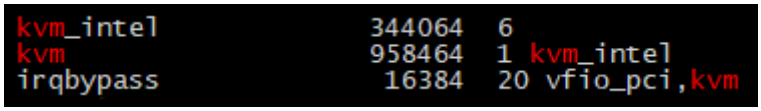
```
# LOAD THE EPEL REPO AND INSTALL THE KVM APPLICATION AND TOOLS
dnf install -y epel-release
dnf install -y libvirt virt-install libvirt-client libguestfs-tools libosinfo virt-top

systemctl enable libvirtd
systemctl start libvirtd
```

Insure the KVM modules got loaded

```
lsmod | grep kvm
```

Should look something like this



Optional:

If you are running a desktop environment, you could install "virt-manager" and manage the KVM with a GUI.

```
dnf install virt-manager
```

Reference: KVM Networks

Virtual Machine Network Configuration (Install)

When creating a virtual machine on the CLI or if like me you create small script files and use `virt-install`,

```
virt-install \
--name sweetname \
--description "Awesome Rocky Server" \
--ram 16384 \
--disk path=/vsto/machines/sweetname.img,size=120 \
--cpu host \
--vcpus=4 \
--cpuset=auto \
--os-variant=rocky8.6 \
--accelerate \
--network bridge:br0 \
--graphics vnc,listen=0.0.0.0 \
--video virtio \
--cdrom /sto/vm/images/rocky-minimal.iso
```

The network line "`--network bridge:br0`" should be modified to use the new OVS bridge networks. From my testing if your configuring for a OVS bridge network that was setup on a access port (untagged port), then it does not appear to matter if you identify the type as either `bridge` or `network`. For example,

```
# BRIDGE
--network bridge:ovs1


# NETWORK
--network network:ovs1
```

The resulting XML that is created for the virtual machine (or 'domain' as they call it), does reflect your choice. So your 'interface' section of the XML could look like these examples,

```
# BRIDGE
   <interface type='bridge'>
     <source bridge='ovs1'/>
   </interface>

# NETWORK
   <interface type='network'>
     <source network='ovs1'/>
   </interface>
```

Either method appears to work.

However, when using a OVS bridge network that was setup on a trunk port (tagged port). We do need to use the `network` type, because we also need to identify the `portgroup` of the VLANs that were defined. In this case we would configure like so,

```
# NETWORK WITH VLAN
--network network:ovs3,portgroup=ext0
```

The XML for this configuration would look like this,

```
# NETWORK WITH VLAN
   <interface type='network'>
     <source network='ovs3' portgroup='ext0'/>
   </interface>
```

This configuration will put your machine on the specified VLAN.

However again, you can also define the VLAN networks within KVM, so you can just call the VLAN name. The configurations page shows both options being used.

Another option that can be added to the network configure line (there is actually a lot that can be configured, but for the sake of this write up, I am sticking to options that pertain to OVS and will make things work) is `virtualport_type=openvswitch`.
This option specifically tells the KVM system that this port is part of the OVS network. I am not sure of all the implications for using this option or not using it. However, I do like the idea of telling the configuration that it should use the OVS networks, and not something that could be configured elsewhere. This adds the following to the XML,

```
# ACCESS PORT (UNTAGGED) NETWORK
<interface type='network'>
```

```
    <source network='ovs1'/>
    <virtualport type='openvswitch'>
      <parameters interfaceid='xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'/>
    </virtualport>
  </interface>

# TRUNK (TAGGED) NETWORK WITH VLAN (PORTGROUP)
  <interface type='network'>
    <source network='ovs3' portgroup='ext0'/>
    <virtualport type='openvswitch'>
      <parameters interfaceid='xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'/>
    </virtualport>
  </interface>
```

For me, I prefer to use the `network` type for all my configurations and I also tag on the `virtualport_type`. This just allows for consistency (at least until I deep dive and fully understand all options and their purposes).

To sum up the install configuration for OVS networks, the following options work,

```
# ACCESS PORT (UNTAGGED) NETWORK
--network network:ovs1,virtualport_type=openvswitch

# TRUNK (TAGGED) NETWORK WITH VLAN (PORTGROUP)
--network network:ovs3,portgroup=ext0,virtualport_type=openvswitch
```

Virtual Machine Network Configuration (Post Install)

Above I have already shown examples of the XML that is created when creating a virtual machine. Here I will show how to modify those setting after a virtual machine is created or for existing machines that will be added to a OVS network.

> BACKUP WARNING!
> Create a backup of your virtual machines XML file before making any changes. Just in case something goes sideways.
> `virsh dumpxml vm-name > vm-name.xml`

> TIME SAVER TIP!
> Before shutting down your VM and changing your network settings within the XML. Change

> the IP information on your VM first so after saving your changes and starting your machine, it will come up on the new network.

First off you will want the virtual machine turned off. The XML of virtual machines are live and get edits when the machines are turned on. So to make a configuration stick (persistent), you will want to shut the machine down.

```
# GRACEFUL SHUTDOWN
virsh shutdown <vm name>

# FORCE SHUTDOWN
virsh destroy <vm name>
```

After the machine is shutdown we can edit the machines XML with the following,

```
virsh edit <vm name>
```

The editor will use what is set as your systems default editor, or will default to using the VI editor.

Comb down through the XML config until you locate the `<interface>` element.

If the machine used a bridge that was setup prior to having OVS, the configuration may look like the following,

```
<interface type='bridge'>
  <mac address='xx:xx:xx:xx:xx:xx'/>
  <source bridge='br0'/>
  <model type='virtio'/>
  <address type='pci' domain='0x0000' bus='0x01' slot='0x00' function='0x0'/>
</interface>
```

We will want to modify the `interface type` within the element, the `source`, and also add the `virtualport type`. Like so,

```
<interface type='network'>
  <mac address='xx:xx:xx:xx:xx:xx'/>
  <source network='ovs1'/>
  <virtualport type='openvswitch'/>
  <model type='virtio'/>
  <address type='pci' domain='0x0000' bus='0x01' slot='0x00' function='0x0'/>
</interface>
```

This config example is using the "ovs1" network from The Crux write up. This network is connected to a access port (untagged), on my switch. So it does not require adding a `portgroup` (vlan) to the configuration.

Changing the network to a trunk port (tagged) and specifying the vlan (portgroup), is done the same way. We just need to add the portgroup to the `source` element. This example will use the "ovs3" network from The Crux write up, and the "ext0" vlan.

```
<interface type='network'>
  <mac address='xx:xx:xx:xx:xx:xx'/>
  <source network='ovs3' portgroup='ext0'/>
  <virtualport type='openvswitch'/>
  <model type='virtio'/>
  <address type='pci' domain='0x0000' bus='0x01' slot='0x00' function='0x0'/>
</interface>
```

Once that is complete, save the XML and turn your machine back on,

```
virsh start <vm name>
```

That's it!

Hope this helps, and perhaps adds a little knowledge to your arsenal. :)