KVM on server with libvirt

This is a work in progress, 20201015

Location of Files

Just a note, the files created by virsh and virt-install are stored in **/etc/libvirt/qemu/**. However, they should not be hand-edited; only edit with virsh.

Install and Configure

First, verify you have the required hardware virtualization support in your CPU:

```
#Verify the hvm support
egrep --color 'vmx|svm' /proc/cpuinfo
```

You should see either vmx or svm in the output.

Now, install the basic packages needed, a couple of utilities, but not all the extra crud.

```
apt install -y --no-install-recommends qemu-kvm libvirt-clients libvirt-
daemon-system bridge-utils libguestfs-tools genisoimage virtinst libosinfo-
bin virt-top
reboot # brings libraries online
```

Verify system is working ok

Commands are of the form: *virsh --connect qemu:///system* ***command*** where *command* is any of the commands accepted by virsh.

```
sudo su # become root user
virsh --connect qemu:///system list --all
```

The *--connect qemu:///system* is the connection used, which is not the default. To set that to the default, run the following one time. This will become the “norm” on the next login:

```
echo "# set default uri for libvirt" >> ~/.profile
echo "export LIBVIRT_DEFAULT_URI='qemu:///system'" >> ~/.profile
```

Defining Network

Setting up bridges

For your network, you need bridges for the outside world.

Simple

This is a basic setup that will work for a single interface as per the Debian documentation. It sets up one bridge off of eth0 and gives it a static IP.

interfaces

```
auto lo
iface lo inet loopback

# The primary network interface
auto eth0

#make sure we don't get addresses on our raw device
iface eth0 inet manual
iface eth0 inet6 manual

#set up bridge and give it a static ip
auto br0
iface br0 inet static
        address 192.168.1.2
        netmask 255.255.255.0
        network 192.168.1.0
        broadcast 192.168.1.255
        gateway 192.168.1.1
        bridge_ports eth0
        bridge_stp off
        bridge_fd 0
        bridge_maxwait 0
        dns-nameservers 8.8.8.8
```

Real World

I'm hoping, if you're reading this article, you know how to set up bonding and vlans. The following is a more real world scenario and sets up three bridges, br_lan, br_dmz and br_wan for LAN, DMZ and public interface respectively. br_wan and br_dmz are set to dummy IP's with a netmask of 255.255.255.255, meaning they can't respond to anything but themselves. br_lan gets its IP from DHCP, so only the LAN interface can be accessed on the hypervisor. Modify it as you want.

interfaces

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

iface eth0 inet manual
iface eth0 inet6 manual

iface eth1 inet manual
iface eth1 inet6 manual

auto bond0
iface bond0 inet manual
   bond-mode 4
   bond-miimon 100
   bond_xit_hash_policy layer2+3
   bond_lacp_rate slow
   slaves eth0 eth1


iface bond0.10 inet manual
   vlan-raw-device bond0.10

iface bond0.20 inet manual
   vlan-raw-device bond0.20

iface bond0.30 inet manual
   vlan-raw-device bond0.30

# the public interface on vlan 10
auto br_wan
iface br_wan inet static
        address 192.168.1.13
        netmask 255.255.255.255
   bridge_ports bond0.10
   bridge_stp off
   bridge_fd 0
   bridge_maxwait 0

# the DMZ on vlan 20
auto br_dmz
iface br_dmz inet static
        address 192.168.1.12
        netmask 255.255.255.255
        bridge_ports bond0.20
        bridge_stp off
        bridge_fd 0
```

```
        bridge_maxwait 0

# the private (LAN) interface on vlan 30
auto br_lan
iface br_lan inet dhcp
        bridge_ports bond0.30
        bridge_stp off
        bridge_fd 0
        bridge_maxwait 0
```

Adding network to virt-lib

In order to use a network with vir-lib, you need to define it. The best way is to create a few XML files, then use virsh to define them into the system.

One at a time

Create one XML file per interface as follows:

br_wan.xml

```
<network>
  <name>br_wan</name>
  <forward mode="bridge"/>
  <bridge name="br_wan"/>
</network>
```

Then, import it into the system with virsh, then set it to autostart on boot

```
# import the network xml file
virsh net-define --file br_wan.xml
# set to autostart on boot
virsh net-autostart br_wan
```

Lazy Approach

I'm lazy, so I just created all three, then imported them all at one time.

br_wan.xml

```
<network>
  <name>br_wan</name>
  <forward mode="bridge"/>
  <bridge name="br_wan"/>
```

```
</network>
```

br_dmz.xml

```
<network>
  <name>br_dmz</name>
  <forward mode="bridge"/>
  <bridge name="br_dmz"/>
</network>
```

br_lan.xml

```
<network>
  <name>br_lan</name>
  <forward mode="bridge"/>
  <bridge name="br_lan"/>
</network>
```

And imported them all at once.

```
for interface in `ls *.xml` ; do virsh net-define --file $interface ; done
for interface in `ls *.xml | cut -d'.' -f1` ; do virsh net-autostart
$interface ; virsh net-start $interface ; done
virsh net-list
```

After the last command, you should see your three interfaces defined. That means you can now use them.

Using Storage

In our example, we are going to use LVM2 to grab a piece of the disk for a new virtual. You can also use a file (File Backed Device, or FBD) by running *fallocate*, or set up access to an iSCSI. virt-install will, by default, create an FBD in its default location, so if you're happy with that, ignore this whole section.

File Backed Device

By default, libvirt uses File Backed Devices (FBD's) from a pool defined internally. You can manually override this by defining a new pool, or by creating a file in your location using fallocate. For example, to create a 10G file in /srv/images named test.disk, you would use:

```
fallocate -l 10G /srv/images/test.disk
```

and use that when you create the virtual. However, it is better (easier) if you define a pool (or use the

default). I have not researched this; see *man virt-install*.

LVM2

Just create an LV the way you always do.

```
lvcreate -L 10G -n test.disk vg0
```

iSCSI

Ok, if you're using iSCSI, I'm guessing you know how to set it up. Just make sure it is available, then use the correct path when you create the image.

Doing the Install

You can probably create a virtual by manually creating the XML file, but why do that when virt-install is your friend. Sure, there are a bunch of parameters, but they are very, very well documented, and will create your system for your rapidly

This example creates a virtual installing the opnSense firewall/router.

```
virt-install                    \
   --hvm                        \
   --connect    qemu:///system \
   --name       router-a       \
   --memory     4096           \
   --vcpus      4              \
   --disk       path=/dev/vg0/router-a.disk0,bus=scsi,target=sda
\
   --graphics vnc,port=5901                                        \
   --noautoconsole                                                 \
   --cdrom     /media/xen-store/OPNsense-20.1-OpenSSL-dvd-amd64.iso \
   --os-variant freebsd11.1                                           \
   --metadata uuid=d9510e01-e461-461f-9aa8-3cee223cb4a0,name=router-
a,title=router-a,description='Primary Router' \
   --boot     hd,cdrom,menu=on                                     \
   --network  bridge=br_wan,mac=00:16:3e:bd:26:70                     \
   --network  bridge=br_dmz,mac=00:16:3e:bd:26:71                  \
   --network  bridge=br_lan,mac=00:16:3e:bd:26:72
```

Other useful options are

- –dry-run # don't really do it
- –controller virtio-scsi # this is specific to KVM, but faster there
- osinfo-query os # this shows the OS's available for the –os-variant flag

most of the parameters are self evident, but I'll quickly talk about why I did some of them.

- –graphics vnc,port=5901 - Our servers are headless and have no GUI. I want to be able to connect during install using VNC. By defining the port here, it is not auto-selected (and changeable) across boots
- –noautoconsole - When starting the virtual, virt-lib tries to bring up a GUI for virtviewer to take over, or run virsh console to bring up a serial console. Since I don't want either, I disable it here
- –os-variant freebsd11.1 - By defining this, virt-install can set up some defaults that work well with the known OS.
- –metadata uuid=d9510e01-e461-461f-9aa8-3cee223cb4a0,name=router-a,title=router-a,description='Primary Router'
    - title, name and description will show up in some reports.
    - uuid is good so the virtual does not get a new, randomly generated UUID every time it runs. Use the uuidgen command to get a randomly generated one for each new machine

Performing Installation

For most systems, I use VNC to do the install. Since I'm remote, I use the following ssh command:

```
ssh -L localhost:5910:localhost:5901 server
```

where

- 5910 is the local VNC port I'll attach to (ie, localhost:5910)
- 5901 is the VNC port I told virt-install to use
- server is the name of the hypervisor

Then, when I run the virt-install command, I simply make a VNC connection to localhost:5910 and can do my install.

Links

- https://www.cyberciti.biz/faq/install-kvm-server-debian-linux-9-headless-server/
- https://www.techotopia.com/index.php/Managing_KVM_on_RHEL_6_using_the_virsh_Command-line_Tool
- https://computingforgeeks.com/virsh-commands-cheatsheet/
- https://serverfault.com/questions/350806/convert-libvirt-xen-configuration-to-native
- https://libvirt.org/sources/virshcmdref/html/
- https://www.utappia.org/2016/04/how-to-migrate-your-virtual-box.html (migrate virtualbox images to kvm qcow2)
- https://serverfault.com/questions/627238/kvm-libvirt-how-to-configure-static-guest-ip-addresses-on-the-virtualisation-ho#627245 (set sticky IP's on KVM)