KVM on server with libvirt

This is a work in progress, 20201015

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.