How to Setup Bridge Networking with KVM on Ubuntu 20.04

How to get your VMs seen by your home network.

Randal Kamradt Sr Follow
May 14, 2020 · 7 min read ★

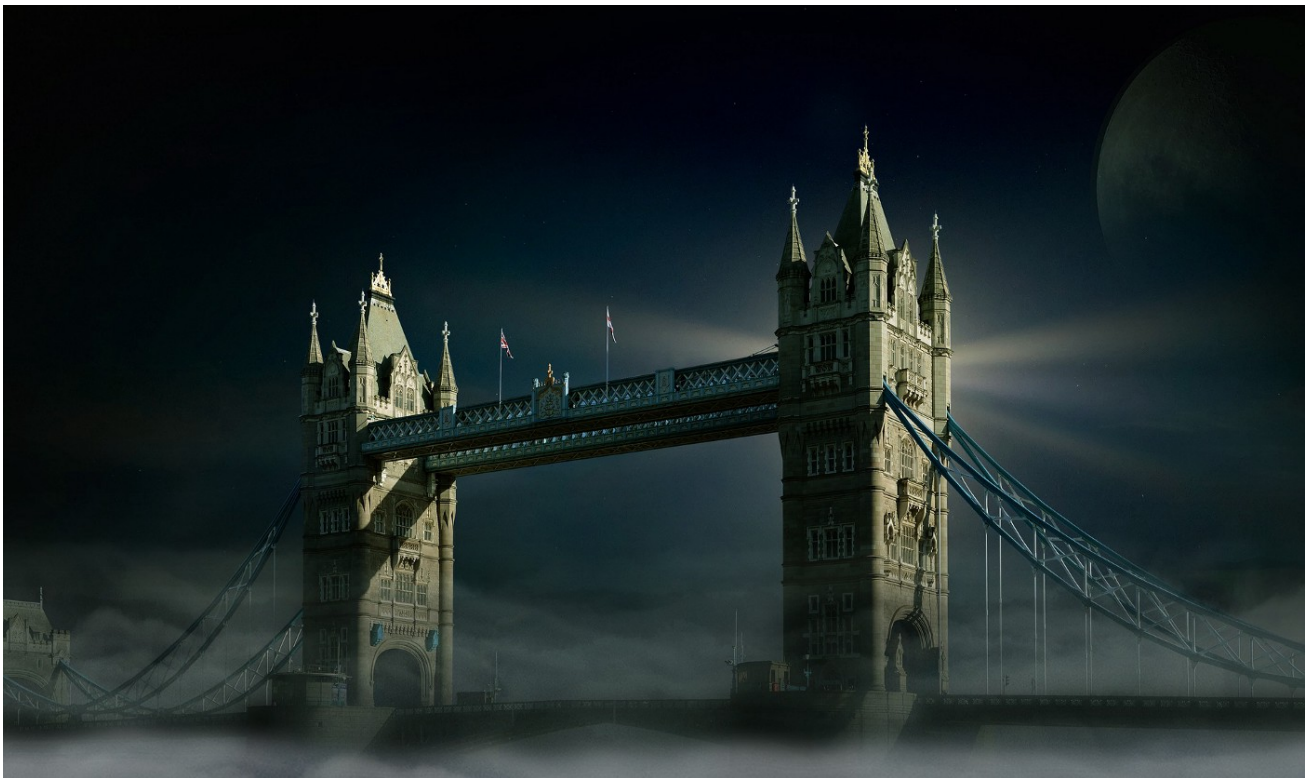

Image by PIRO4D from Pixabay

The default networking that KVM provides is a private bridge that uses NAT to allow the guest VMs to talk to the outside world. This works for a lot of use-cases and is probably the best default setup. But sometimes I need the VMs to share the network with the host. In many of my past articles that created VMs, I've been able to get

away with the default networking. But a future project I'm working on will need VMs operating on the host network, so I wanted to document how to do that here.

There are a lot of articles on setting up a public bridge; it seems like a popular topic. But many of the articles are old and don't use the latest networking configuration software available with Ubuntu or they manage networking and KVM with a GUI interface. I will be using only command-line tools, as I don't actually have monitors for my servers.

So here is my take on setting up KVM to use a public bridge by default. To follow along, you should have good knowledge of Linux, KVM, and modern Linux networking. We will be using Ubuntu 20.04 which uses the `iproute2` package for command-line access, and `netplan` for configuration. Ubuntu 20.04 was released in April 2020 (hence the version number 20.04) but I have been using the daily builds since the beginning of the year to get familiar with it.

First, I'm going to install KVM, the virtual machine manager. For Ubuntu 20.04 The installation is easy:

```
sudo apt-get install qemu-kvm libvirt-daemon-system \
   libvirt-clients virtinst bridge-utils
```

Test that it's working with `virsh list --all` this should show you an empty listing, but a listing none-the-less:

```
rkamradt@artful:~$ virsh list --all
 Id   Name   State
--------------------
```

The next thing we're going to do is replace the default bridging. As mentioned above, KVM installs a virtual bridge that all the VMs connect to. It provides its own subnet and DHCP to configure the guest's network and uses NAT to access the outside world. We're going to replace that with a public bridge that runs on the host network and uses whatever external DHCP server is on the host network.

For performance reasons, it is recommended to disable `netfilter` on bridges in the

host. To do that, create a file called `/etc/sysctl.d/bridge.conf` and fill it in with this:

```
net.bridge.bridge-nf-call-ip6tables=0
net.bridge.bridge-nf-call-iptables=0
net.bridge.bridge-nf-call-arptables=0
```

Then create a file called `/etc/udev/rules.d/99-bridge.rules` and add this line:

```
ACTION=="add", SUBSYSTEM=="module", KERNEL=="br_netfilter", \
RUN+="/sbin/sysctl -p /etc/sysctl.d/bridge.conf"
```

That should all be on one line. That will set the flags to disable `netfilter` on bridges at the proper place in system start-up. Reboot to take effect.

Next, we need to disable the default networking that KVM installed for itself. You can use `ip` to see what the default network looks like:

```
rkamradt@beast:~$ ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state
UP mode DEFAULT group default qlen 1000
    link/ether d4:be:d9:f3:1e:5f brd ff:ff:ff:ff:ff:ff
6: virbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
noqueue state UP mode DEFAULT group default qlen 1000
    link/ether 52:54:00:1d:5b:25 brd ff:ff:ff:ff:ff:ff
7: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc fq_codel
master virbr0 state DOWN mode DEFAULT group default qlen 1000
    link/ether 52:54:00:1d:5b:25 brd ff:ff:ff:ff:ff:ff
```

The entries `virbr0` and `virbr0-nic` are what KVM installs by default.

Because this host just had KVM installed, I don't have to worry about existing VMs. If you have existing VMs, you will have to edit them to use the new network setup. It's possible you can have both public and private VMs, but I'd just as soon have only one type of network per host to avoid confusion. Here's how to remove the default KVM network:

```
virsh net-destroy default
virsh net-undefine default
```

Now you can run `ip` again and the `virbr0` and `virbr0-nic` should be gone.

```
rkamradt@beast:~$ ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state
UP mode DEFAULT group default qlen 1000
    link/ether d4:be:d9:f3:1e:5f brd ff:ff:ff:ff:ff:ff
```

If they are not, you can remove them with `ip link delete virbr0 type brigde` and `ip link delete virbr0-nic`.

Next, we will need to set up a bridge to use when we create a VM. Edit your `/etc/netplan/00-installer-config.yaml` (after making a back-up) to add a bridge. This is what mine looks like after editing:

```
network:
  ethernets:
    enp0s7:
      dhcp4: false
      dhcp6: false
  bridges:
    br0:
      interfaces: [ enp0s7 ]
      addresses: [192.168.0.104/24]
      gateway4: 192.168.0.1
      mtu: 1500
      nameservers:
        addresses: [8.8.8.8,8.8.4.4]
      parameters:
        stp: true
        forward-delay: 4
      dhcp4: no
      dhcp6: no
  version: 2
```

In my case `enp0s7` is the name of my NIC, `192.168.0.104` is the IP address of my host, and `192.168.0.1` is my home router. I've set up my home router to reserve `192.168.0.104` for this host by associating the MAC address to the IP address. The bridge `br0` is attached to the `enp0s7` interface, the physical network card on the host. Notice that the `enp0s7` interface is not configured, the bridge now has the network configuration that used to be specified in the `enp0s7` section. I'm not sure about the `parameters` section, I copied them straight from an example setup and I may want to play with them later.

Now run `sudo netplan apply` to apply your new configuration. You can use the `ip` command to inspect that it looks correct:

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
fq_codel master br0 state UP group default qlen 1000
    link/ether 00:1d:72:ac:d9:95 brd ff:ff:ff:ff:ff:ff
13: br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
state UP group default qlen 1000
    link/ether 00:1d:72:ac:d9:95 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.104/24 brd 192.168.0.255 scope global br0
       valid_lft forever preferred_lft forever
    inet6 fe80::21d:72ff:feac:d995/64 scope link
       valid_lft forever preferred_lft forever
```

Note that the `br0` entry now has the IP address and the `enp0s7` entry now has `master br0` to show that it belongs to the bridge.

Now we can make KVM aware of this bridge. create a scratch XML file called `host-bridge.xml` and insert the following:

```
<network>
  <name>host-bridge</name>
  <forward mode="bridge"/>
```

```
  <bridge name="br0"/>
</network>
```

Use the following commands to make that our default bridge for VMs:

```
virsh net-define host-bridge.xml
virsh net-start host-bridge
virsh net-autostart host-bridge
```

And then list the networks to confirm it is set to autostart:

```
rkamradt@artful:~$ virsh net-list --all
 Name          State    Autostart   Persistent
----------------------------------------------
 host-bridge   active   yes         yes
```

Now that we have a bridge configured, we can make a VM. Here is the template I use:

```
1  virt-install --name vm1 --ram=8192 --disk size=10 --vcpus 1 --os-type linux --os-vari
2  # When prompted, use the vm name for the host name, and be sure to add openssh when a
3  # otherwise take defaults or most reasonable options. Follow install instructions til
4  virsh domifaddr vm1 # get ip address
5  ssh-copy-id rkamradt@192.168.122.95 # use ip address from previous step
```

installVM.sh hosted with ❤ by **GitHub** **view raw**

Once the `virt-install` program is running, you'll see the installation program running in your terminal. Install it however you like, but make sure you install OpenSSH when it prompts for additional software to install as that is what we'll use to communicate with our new VM. Now have a cup of coffee while you wait for the installation.

One thing that was inconvenient was that it configured the VM's network with DHCP instead of asking if you want to do it manually. With servers, it's nice to configure manually with a static IP address drawn from outside DHCP's pool. Even the `virsh` tool is unaware of the new guest's network.

```
rkamradt@artful:~$ virsh domifaddr node1
 Name       MAC address          Protocol     Address
-------------------------------------------------------------------------------
--
```

The only way I could find out the address was to go on my home router and look for `node1`. My home router tries to keep IP addresses the same based on MAC addresses, but it's not guaranteed, so I added it to the list of static IPs. Each router is different, so you'll have to figure out how to do that with your own router. Add the new VM to the `/etc/hosts` file on whatever hosts you plan to access it on, so you can refer to it by name instead of remembering the IP address.

Use the `ip` command again to view your network on the host:

```
rkamradt@artful:~$ ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp0s7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
fq_codel master br0 state UP mode DEFAULT group default qlen 1000
    link/ether 00:1d:72:ac:d9:95 brd ff:ff:ff:ff:ff:ff
13: br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
state UP mode DEFAULT group default qlen 1000
    link/ether 00:1d:72:ac:d9:95 brd ff:ff:ff:ff:ff:ff
15: vnet0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
fq_codel master br0 state UNKNOWN mode DEFAULT group default qlen
1000
    link/ether fe:54:00:a3:0d:ee brd ff:ff:ff:ff:ff:ff
```

It added `vnet0` to the network. For each VM you install, it adds a new `vnet#` I'm not exactly sure why, and I hope I've done this right, but it seems to work.

Once your VM is configured, you'll have to pass your ssh credentials to it. The easiest way is with `ssh-copy-id <username>@<guestname>` If your username is the same between the host computer and the guest computer, you don't need the `<username>@` part. Then you should be able to connect with `ssh <username>@<guestname>`.

Once in the guest, you can use the `ip` command to view the network:

```
rkamradt@node1:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
fq_codel state UP group default qlen 1000
    link/ether 52:54:00:a3:0d:ee brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.166/24 brd 192.168.0.255 scope global enp1s0
       valid_lft forever preferred_lft forever
    inet6 fe80::5054:ff:fea3:dee/64 scope link
       valid_lft forever preferred_lft forever
```

I used the `ip addr` subcommand this time to see the IP address instead of just the MAC addresses. Notice that the IP address is in the same network as the host, in this case `192.168.0.0/24` You should be able to access it anywhere in the network.

I'm not a network expert by any means, but I can usually struggle through getting things to work. Hopefully, this will help you get a handle on networking and the new generation of commands and configurations. If you're only working on cloud-provider VMs much of this is done for you with (hopefully) easy UIs. But if you're just a hobbyist like me wanting to create VMs on bare-metal to try out different software, and have the ease of bringing them up and taking them down at will, it's

Thanks to Zack Shapiro.

Ubuntu Kvm Network