Custom NAT-based network

The default *NAT-based network* has some limitations. Follow the steps below to overcome these limitations and take control of your server environment. There are four main components: a dummy network interface, a virtual bridge, some iptables rules, and dnsmasq.

> **Note**
>
> If you are uncomfortable with iptables, you might prefer to stick with the default *NAT-based network*.

Disable the default network

To prevent libvirt from altering the firewall, stop and disable the default network. Make sure there are no active virtual machines (VMs) still using this network.

```
# virsh net-destroy default
# virsh net-autostart --disable default
```

Create a dummy interface

A bridge inherits the MAC address of the first interface that is attached, so it will keep changing unless the same VM is always powered on first. To keep the MAC address constant, create a dummy network interface with a chosen MAC address and attach it to the bridge before anything else.

Choose a MAC address for the virtual bridge. Use `hexdump` to generate a random MAC address in the format that libvirt expects (`52:54:00:xx:xx:xx` for KVM, `00:16:3e:xx:xx:xx` for Xen).

```
# hexdump -vn3 -e '/3 "52:54:00"' -e '/1 ":%02x"' -e '"\n"' /dev/urandom
52:54:00:7e:27:af
```

Create a dummy network interface called `virbr10-dummy` and set the MAC address to the one generated above.

```
# ip link add virbr10-dummy address 52:54:00:7e:27:af type dummy
```

To create a persistent dummy interface at every boot, follow the instructions for *Red Hat Enterprise Linux*, *Fedora*, or *Debian*.

Create a virtual bridge

A Linux bridge without a real Ethernet device is considered virtual. Choose a name and a private subnet for the virtual bridge. In this example:

- The virtual bridge is called `virbr10`.
- The private subnet chosen is `192.168.100.0/24` (see CIDR notation).
  - VMs can bind to addresses from `192.168.100.2` to `192.168.100.254`.
  - VMs see the libvirt server as `192.168.100.1`.

To manually create the virtual bridge, run these commands:

```
# brctl addbr virbr10
# brctl stp virbr10 on
# brctl addif virbr10 virbr10-dummy
# ip address add 192.168.100.1/24 dev virbr10 broadcast 192.168.100.255
```

Rather than creating the bridge manually, follow the instructions for *Red Hat Enterprise Linux*, *Fedora*, or *Debian* (so that the bridge is created at every boot).

Implement NAT with iptables

IP masquerading allows many machines with private IP addresses (eg, `192.168.X.X`) to communicate with the Internet through the public IP address of a router. In this case, the libvirt server acts as a router for the VMs.

A router must be able to forward network packets between interfaces, so modify some kernel parameters to allow this.

```
# echo "net.ipv4.ip_forward=1" >> /etc/sysctl.conf
# echo "net.ipv4.conf.all.forwarding=1" >> /etc/sysctl.conf
# sysctl -p
```

Implement IP masquerading in the `nat` table.

```
# This format is understood by iptables-restore. See `man iptables-restore`.
*nat
:PREROUTING ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
# Do not masquerade to these reserved address blocks.
-A POSTROUTING -s 192.168.100.0/24 -d 224.0.0.0/24 -j RETURN
-A POSTROUTING -s 192.168.100.0/24 -d 255.255.255.255/32 -j RETURN
# Masquerade all packets going from VMs to the LAN/Internet.
-A POSTROUTING -s 192.168.100.0/24 ! -d 192.168.100.0/24 -p tcp -j MASQUERADE --
```

```
-A POSTROUTING -s 192.168.100.0/24 ! -d 192.168.100.0/24 -p udp -j MASQUERADE --
-A POSTROUTING -s 192.168.100.0/24 ! -d 192.168.100.0/24 -j MASQUERADE
COMMIT
```

Allow forwarding in the `filter` table.

```
# This format is understood by iptables-restore. See `man iptables-restore`.
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]

... snipped ...
# Allow established traffic to the private subnet.
-A FORWARD -d 192.168.100.0/24 -o virbr10 -m conntrack --ctstate RELATED,ESTABLI
# Allow outbound traffic from the private subnet.
-A FORWARD -s 192.168.100.0/24 -i virbr10 -j ACCEPT
# Allow traffic between virtual machines.
-A FORWARD -i virbr10 -o virbr10 -j ACCEPT
# Reject everything else.
-A FORWARD -i virbr10 -j REJECT --reject-with icmp-port-unreachable
-A FORWARD -o virbr10 -j REJECT --reject-with icmp-port-unreachable
... snipped ...
COMMIT
```

See *Example of iptables NAT* for a full iptables rule set.

Run dnsmasq

> **❗ Note**
>
> This step is optional, as VMs can bind to addresses without DHCP and can use their own DNS resolver. However, it is recommended unless you know what you are doing.

On the libvirt server, it is common to run a DHCP server (to decide which IP address to lease to each VM) and a DNS server (to respond to queries from VMs). dnsmasq is ideal for both of these tasks.

Create files and directories needed for dnsmasq.

```
# mkdir -p /var/lib/dnsmasq/virbr10
# touch /var/lib/dnsmasq/virbr10/hostsfile
# touch /var/lib/dnsmasq/virbr10/leases
```

Create `/var/lib/dnsmasq/virbr10/dnsmasq.conf` with these contents:

```
# Only bind to the virtual bridge. This avoids conflicts with other running
# dnsmasq instances.
except-interface=lo
interface=virbr10
bind-dynamic

# If using dnsmasq 2.62 or older, remove "bind-dynamic" and "interface" lines
# and uncomment these lines instead:
#bind-interfaces
#listen-address=192.168.100.1

# IPv4 addresses to offer to VMs. This should match the chosen subnet.
dhcp-range=192.168.100.2,192.168.100.254

# Set this to at least the total number of addresses in DHCP-enabled subnets.
dhcp-lease-max=1000

# File to write DHCP lease information to.
dhcp-leasefile=/var/lib/dnsmasq/virbr10/leases
# File to read DHCP host information from.
dhcp-hostsfile=/var/lib/dnsmasq/virbr10/hostsfile
# Avoid problems with old or broken clients.
dhcp-no-override
```

```
# https://www.redhat.com/archives/libvir-list/2010-March/msg00038.html
strict-order
```

Optionally, use `hostsfile` to always assign a specific IP address to a VM with a specific MAC address. (dnsmasq only reads changes after receiving a `SIGHUP`.)

```
# echo "52:54:00:be:0a:f3,192.168.100.77" \
      >> /var/lib/dnsmasq/virbr10/hostsfile
```

Add some iptables rules. (See *Example of iptables NAT* for a full iptables rule set.)

```
*filter
... snipped ...
# Accept DNS (port 53) and DHCP (port 67) packets from VMs.
-A INPUT -i virbr10 -p udp -m udp -m multiport --dports 53,67 -j ACCEPT
-A INPUT -i virbr10 -p tcp -m tcp -m multiport --dports 53,67 -j ACCEPT
... snipped ...
COMMIT

*mangle
:PREROUTING ACCEPT [0:0]
```

```
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
# DHCP packets sent to VMs have no checksum (due to a longstanding bug).
-A POSTROUTING -o virbr10 -p udp -m udp --dport 68 -j CHECKSUM --checksum-fill
COMMIT
```

If you are running a system-wide instance of dnsmasq, you may need to configure it to ignore the virtual bridge.

```
# touch /etc/dnsmasq.d/virbr10.conf
# echo "except-interface=virbr10" >> /etc/dnsmasq.d/virbr10.conf
# echo "bind-interfaces" >> /etc/dnsmasq.d/virbr10.conf
# service dnsmasq restart
```

If systemd is available, *Run dnsmasq with systemd*. Otherwise, just run dnsmasq from the command-line:

```
# dnsmasq --conf-file=/var/lib/dnsmasq/virbr10/dnsmasq.conf \
      --pid-file=/var/run/dnsmasq/virbr10.pid
```

Forward incoming connections

> **Note**
>
> This step is optional. It is only necessary if one or more VMs are running services (eg, web applications) that need to be available over the network.

If one of the VMs has a web application listening on ports `80`/`443`, connections to those ports on the server (eg, `203.0.113.3`) can be forwarded to the VM (eg, `192.168.100.77`). Add these iptables rules:

```
*nat
... snipped ...
# Modify the destination address of packets received on ports 80 and 443.
```

```
-A PREROUTING -d 203.0.113.3/32 -p tcp -m tcp --syn -m multiport --dports 80,443

# Optionally, make the VM accessible via `ssh -p 2222 user@203.0.113.3`.
-A PREROUTING -d 203.0.113.3/32 -p tcp -m tcp --syn --dport 2222 -j DNAT --to-de
COMMIT

*filter
... snipped ...
# Allow packets that have been forwarded to particular ports on the VM.
-A FORWARD -d 192.168.100.77/32 -o virbr10 -p tcp -m tcp --syn -m conntrack --ct
... snipped ...
COMMIT
```

The main limitation is that a specific port on the server can only be forwarded to a single VM. This is problematic if many VMs are fighting over ports `80`/`443`. One option is to forward connections to ports `80`/`443` on the server to a VM running a reverse proxy (eg, NGINX or HAProxy), which can then proxy those connections to other VMs. Alternatively, run a reverse proxy on the libvirt server itself.

See *Example of iptables NAT with connection forwarding* for a full iptables rule set.

Configure virtual machines

New VM

```
# virt-install --network bridge=virbr10 ...
```

Optionally, pass `--network` more than once to create additional virtual Ethernet interfaces for the VM.

```
# virt-install --network network=virbr10 --network network=virbr11 ...
```

Existing VM

Open the XML configuration for the VM in a text editor.

```
# virsh edit name-of-vm
```

There should already be an `<interface>` section that configures a virtual Ethernet interface for the VM. Note down the MAC address.

```
<interface type="network">
   <source network="default"/>
   <mac address="52:54:00:4f:47:f2"/>
</interface>
```

To reconfigure the virtual Ethernet interface, replace the `<interface>` section with the following contents. Use the MAC address noted above, otherwise the MAC address of the VM will change.

```
<interface type="bridge">
  <source bridge="virbr10"/>
  <mac address="52:54:00:4f:47:f2"/>
</interface>
```

To add an additional virtual Ethernet interface, append a new `<interface>` section. libvirt generates a random MAC for the new interface if `<mac>` is omitted.

```
<interface type="network">
   <source network="virbr10"/>
</interface>
```

Reboot the VM to apply the changes. You may need to amend the VM’s network initialization scripts to account for the network interface changes.

Comments

⟵ Previous

Next ⟶

Version 1.0.1 — Last updated on 2015-12-16.