Custom routed network

libvirt's built-in *Routed 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 built-in *Routed network*.

Initial steps

In this example:

- The server has an Ethernet device called `eth0`.
- VMs bind to a virtual bridge called `virbr10`.
- The hosting provider has allocated two address blocks (see CIDR notation):
  - one public IPv4 address block (`203.0.113.80/28`)
  - one public IPv6 address block (`2001:db8::/56`)
- The server binds statically to `203.0.113.86` and `2001:db8::1`.

To begin, choose which IP addresses to make available to VMs. In this example, a subnet of six IPv4 addresses (`203.0.113.88/29`) will be sliced from the allocated address block and made available to

VMs. In an Intranet or home lab, you might choose a whole subnet (eg, `192.168.50.0/24`).

For IPv6, always use a `/64` subnet prefix to avoid breaking various IPv6 features. In this example, `2001:db8::/56` can be sliced into 256 different `/64` subnets, one of which (`2001:db8:aa::/64`) will be made available to VMs. (Most hosting providers allocate a `/64` block for every server, and should allocate a free `/56` on request.)

Configure static routes

The LAN router is not yet aware that packets destined for the subnets above need to be routed to the libvirt server. Packets will only reach VMs if you configure static routes on the LAN router. In this example, two static routes are required:

- `ip -4 route add 203.0.113.88/29 via 203.0.113.86`
- `ip -6 route add 2001:db8:aa::/64 via 2001:db8::1`

For a dedicated server, ask your hosting provider to configure these routes for you. For an Intranet or home lab, consult the documentation for your router.

Note

If the router in your home lab is consumer-grade, it might have buggy support for static routes or it might not allow static routes at all. If possible, install alternative firmware such as OpenWRT or Merlin. (A painful last

> resort is to add static routes to every client on the LAN.)

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. To manually create the virtual bridge, run these commands:

```
# brctl addbr virbr10
# brctl stp virbr10 on
# brctl addif virbr10 virbr10-dummy
# ip address add 203.0.113.88/29 dev virbr10 broadcast 203.0.113.95
# ip address add 2001:db8:aa::/64 dev virbr10
```

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).

Configure the firewall

Note

You might want to disable the default network before you continue.

When network packets destined for VMs (`203.0.113.88/29`) arrive at the libvirt server, they are forwarded to the virtual bridge. 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
# echo "net.ipv6.conf.all.forwarding=1" >> /etc/sysctl.conf
# sysctl -p
```

Allow IPv4 forwarding for `iptables`.

```
# 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 inbound traffic to the private subnet.
-A FORWARD -d 203.0.113.88/29 -o virbr10 -j ACCEPT
# Allow outbound traffic from the private subnet.
-A FORWARD -s 203.0.113.88/29 -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
```

Allow IPv6 forwarding for `ip6tables`.

```
# This format is understood by ip6tables-restore. See `man ip6tables-restore`.
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]

... snipped ...
# Allow inbound traffic to the private subnet.
-A FORWARD -d 2001:db8:aa::/64 -o virbr10 -j ACCEPT
# Allow outbound traffic from the private subnet.
-A FORWARD -s 2001:db8:aa::/64 -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 icmp6-port-unreachable
-A FORWARD -o virbr10 -j REJECT --reject-with icmp6-port-unreachable
... snipped ...
COMMIT
```

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
bind-dynamic
interface=virbr10

# If using dnsmasq 2.62 or older, remove "bind-dynamic" and "interface" lines
# and uncomment these lines instead:
#bind-interfaces
#listen-address=203.0.113.88
#listen-address=2001:db8:aa::1

# IPv4 addresses to offer to VMs. This should match the chosen subnet.
dhcp-range=203.0.113.89,203.0.113.94

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

# Assign IPv6 addresses via stateless address autoconfiguration (SLAAC).
dhcp-range=2001:db8:aa::,ra-only

# Assign IPv6 addresses via DHCPv6 instead (requires dnsmasq 2.64 or later).
# Remember to allow all incoming UDP port 546 traffic on the VM.
#dhcp-range=2001:db8:aa::1000,2001:db8:aa::1fff
#enable-ra
#dhcp-lease-max=5000

# 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
```

Note

dnsmasq only supports SLAAC in version `2.64` onwards. If using an older version, see *Run radvd*.

Optionally, use `hostsfile` to always assign the same IPv4 address to a particular VM using its MAC address. dnsmasq only reads changes after receiving a `SIGHUP`.

```
# echo "52:54:00:3b:9a:2b,203.0.113.90" \
      >> /var/lib/dnsmasq/virbr10/hostsfile
# killall -SIGHUP -e dnsmasq
```

Optionally, use `hostsfile` to always assign the same IPv6 address to a particular VM using its DHCP Unique Identifier (DUID). For example, if a VM is currently bound to `2001:db8:aa::1010`, determine the DUID of the VM using the `leases` file and then add a `hostsfile` entry.

```
# awk '$3=="2001:db8:aa::1010" {print $NF}' /var/lib/dnsmasq/virbr10/leases
00:01:00:01:1a:ff:2b:ff:52:54:00:3b:9a:2b

# echo "id:00:01:00:01:1a:ff:2b:ff:52:54:00:3b:9a:2b,[2001:db8:aa::1010]" \
      >> /var/lib/dnsmasq/virbr10/hostsfile
# killall -SIGHUP -e dnsmasq
```

Add some firewall rules for both `iptables` and `ip6tables`.

```
*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

# If using DHCPv6 instead of SLAAC, also allow UDP port 547.
# -A INPUT -i virbr10 -p udp -m udp --dport 547 -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
```

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 →