Setting up your own internal network w/ qemu.
@ Articles -> Networking Aug 20 2007, 15:42 (UTC+0)

bulibuta writes: Setting up your own internal network w/ qemu.

Many times I found myself wanting to test something on a different arch or look at what a new OS brings (specially the hobbyist ones announced now and then on http://www.osnews.com OSNews>) or simply test some network configurations w/o affecting the existing one. So I started looking at virtual emulators and such.

After some documentation and searching qemu seemed to be the best to fit my needs and closest to my philosophy. In this article I'll present a way to quickly build a virtual network environment with http://www.OpenBSD.org OpenBSD> as host.

The first thing to do is, obviously, get qemu. On OpenBSD you can simply do a ``pkg_add qemu'' and that's that. For other platforms check the qemu homepage or your local repository.

Next you should get the ISO's for the operating system you intend to install as guest. (I'll use debian in this case, but any other will have similar installing steps.)

With your ISO and qemu install you need to create a virtual disk for your guest OS.

```
qemu-img create -f qcow debian.hd 4G
```

Now we have a 4 gigabytes image formated with qemu's own ``filesystem''. Don't worry about how big you make it. Qemu will allocate space as you go along and fill it. What you basically do here is set an upper bound for the image size.

With the image ready the installation process is good to go:

```
qemu -m 128 -cdrom debian.iso -boot d -monitor stdio debian.hd
```

This will bring up an instance of qemu that boots from the cdrom image passed as argument, let's the guest OS have 128 megabytes of RAM and drops you inside the qemu console. From inside the console you can change the cdrom image, send key combos etc.

If you don't have a cdrom image but a floppy disk you can do something pretty much the same except that you load the floppy disk and boot from it:

```
qemu -m 128 -floppy floppy.img -boot a -monitor stdio debian.hd
```

After this command is issued you'll get a window with the install process of your selected OS. The steps to install it are no different than the ones of a real install.

Okay. The install process is over, you have a raw OS running as guest, but how do you upgrade, add patches, install new software etc.? You need interaction with the outside world. You can load other cdrom images with the packages you want but that means downloading a hole ISO for a few actually needed packages. And if you want to test certain applications that need network access? Well... everything can be easily done with a bridge, a tap and some nat-ing.

Qemu has a small configuration file that runs every time an instance of qemu is started. On most systems it can be found under /etc/qemu-ifup. There are some examples in your /usr/local/share/doc/qemu directory for it. Mainly in this script the bridge and tap device are initiated.

If you do not know what a bridge or/and a tap device is you can read up about them
http://www.openbsd.org/cgi-bin/man.cgi?query=bridge&apropos=0&sektion=0&manpath=OpenBSD+Current&arch=i386&format=html here> and
http://www.openbsd.org/cgi-bin/man.cgi?query=tun&apropos=0&sektion=0&manpath=OpenBSD+Current&arch=i386&format=html here> (or check your local documentation). The people at qemu have some nice http://fabrice.bellard.free.fr/qemu/user-doc.html documents> too.

Anyway here is a configuration file that should give you a head start:

```
#! /bin/sh
set -x

_ETHER=dc0
```

```
_BRIDGE=bridge0

# Let the environment over-ride this
[ "$BRIDGE" ] || BRIDGE=${_BRIDGE}
[ "$ETHER" ] || ETHER=${_ETHER}

if test `id -u` -ne 0; then
    SUDO=sudo
fi

echo "Initializing $1.."

# Set the tun device into layer2 mode
$SUDO ifconfig $1 link0 up

# Set up our bridge
$SUDO ifconfig $1 group tun
$SUDO ifconfig $1 10.0.0.1 netmask 255.255.255.0
$SUDO ifconfig $BRIDGE create && {
  # Ony add rules if the bridge creation succeeds; otherwise
  # duplicate rules get loaded each time qemu starts
  # The following two block carp packets from wasting cpu cycles inside the
  # qemu sessions, remove if testing carp inside qemu
  $SUDO brconfig $BRIDGE rule block in on $ETHER dst 33:33:0:0:0:12
  $SUDO brconfig $BRIDGE rule block in on $ETHER dst 01:00:5e:00:00:12
}
# Since we can specify ETHER and BRIDGE above, its possible that
# this tun interface or this physical interface was setup as part of
# a different bridge earlier, and that is never cleaned up, so we have
# to cleaup here first before we set it up; a physcal interface cannot
# be member to more than one bridge, thankfully, or I never would have
# caught this
ifconfig bridge | sed -n '/^bridge[0-9]*/{s/:.*$//;p;}' | while read brif
do
    $SUDO brconfig $brif del $ETHER
    $SUDO brconfig $brif del $1
done
$SUDO brconfig $BRIDGE add $ETHER up
$SUDO brconfig $BRIDGE add $1 up || true
```

Much of this file was taken from the examples in the share/doc/qemu directory. I only added the IP and netmask configuration for the tap device and modified my bridge and network interface device (on Linux the dc0 will be eth0 or whatever).

That's it. Now we can start qemu with networking support:

```
qemu -m 128 -monitor stdio -net nic -net tap debian.hd
```

The nic switch emulates a ne2000 network card (by default) inside of the guest OS and the tap switch enables the tap interface. Remember to configure your guest OS with a corresponding netmask and IP (e.g. 255.255.255.0 and 10.0.0.2).

Now you can ping the guest OS (at IP 10.0.0.1) and SSH/SCP stuff to it. But what if you want actual internet access. Nothing to it, you'll just have to add some nating to the host for the packets incoming from your internal qemu network in your pf.conf (iptables for Linux or you're favorite packet filter).

In OpenBSD a pf.conf would look like this (see my former http://neworder.box.sk/news/14689 article> on pf):

```
### CONSTANTS ###

#ethernet interfaces
internal = "tun0"
external = "dc0"

#computers behind

#services running on the server
services = "{ ssh, www, https, ftp }"

icmp_types = "echoreq"

### OPTIONS ###

set block-policy return
set loginterface $external
set skip on lo0

### SCRUB ###

scrub in all

### NAT ###

nat on $external from $internal:network to any -> ($external)

### REDIRECTS TO LOCAL MACHINE ###
```

```
### FILTER RULES ###

  block all

  pass in log on $external inet proto tcp from any to $external
        port $services flags S/SA keep state

  pass in log inet proto icmp all icmp-type $icmp_types keep state

  pass in log quick on $internal from $internal:network to any
        keep state
  pass out log quick on $internal from any to $internal:network
        keep state
  pass out log on $external proto tcp all modulate state flags
        S/SA
  pass out log on $external proto { udp, icmp } all keep state


### EOF ###
```

Now just enable the packet filter and add a default gateway route to your guest OS and you have internet access from your qemu network.

Remember to check the snapshot options for the qemu images (that way you can keep a base install for all and add different things to it like dedicating one for web hosting, one for devel, one for bashing etc.). Another interesting feature is compressing an image after you added pretty much all you wanted to it.

That's it. From here you can play on... If you have any questions or make some interesting things with your config remember to write them in the comments section.
