Virtualizing with KVM in Linux

June 15, 2015
Last updated: November 7, 2017

> *If you do what you've always done, you'll get what you've always gotten.*
> *—Anthony Robbins*

Most of you are familiar with full virtualization solutions like VMware Workstation, Oracle VirtualBox, and Parallels. In this post, I'll re-introduce you to another, arguably faster, way of doing things.

The **$** symbol will be used to indicate the shell prompt for a regular user, while the **#** symbol will denote the shell for the root user. There are cases when the EUID of a command will be zero (0) due to the use of sudo.

Table of contents

- Setup
  - Hardware
  - Software
- Configuration
  - Images
  - Networking
- Execution
  - Load the image
  - Connect to the SPICE display
  - Configure guest networking
- Closing the curtains
  - Restore networking
- Putting it all
- Closing remarks

Setup

Hardware

One of the first things that you need to do is to enable Hardware-assisted virtualization, also called accelerated virtualization, in your hardware. If your CPU was made before 2006, chances are, this feature won't be present on your chip. Also, take note that this step is not compulsory to make use of the virtualization solution described in this post, but it will *significantly* speed things up.

To enable accelerated virtualization, go into your BIOS/UEFI settings, and look for the knob that enables this feature. The name into which it goes is different from manufacturer to manufacturer. Save the settings, then boot your system. You can verify on the command line if your system indeed recognizes it.

```
$ egrep '(vmx|svm)' /proc/cpuinfo
```

If it returns some text, then you're good.

Software

Next, you need to install the essential applications.

Nix:

```
$ nix-env -i qemu vde2 spice_gtk
```

APT:

```
$ sudo apt-get install qemu-kvm vde2 spice-client-gtk
```

This will install the QEMU /kee-MOO/ hypervisor, VDE tools, and SPICE support. QEMU, at least during its early days had the *meh* impression—it is OK, but not stellar. Since version 0.10.1, QEMU started supporting KVM, a virtualization subsystem for Linux, that provides near-native virtualization performance using hardware-assisted virtualization. It even rivals the performance of the virtualization solutions mentioned above.

Other suites offer the option of connecting to the guest machine's display via VNC. The problem is that, it's slow and clunky. The response time is just horrible. Using the SPICE protocol, not only does it makes things faster, but it makes other things possible. Take note that SPICE is not a replacement for VNC, but rather, it a different way of meeting your goals.

Configuration

Images

QEMU supports an array of image types, however the QCOW2 format is the most flexible, and feature-rich, for QEMU use.

If you have an existing VirtualBox file (VDI), you can convert it to QCOW2 with:

```
$ qemu-img convert -f vdi -O qcow2 vm.vdi vm.qcow2
```

However, if you don't have an image, yet, you can create one with:

```
$ qemu-img create -f qcow2 vm.qcow2 20G
```

The last step creates a 20GB image, that is named **`vm.qcow2`**. Take note that the extension name doesn't really matter—you can name your image as **`index.html`**, but that wouldn't make a lot of sense, right? 😊

KVM group

The commands below require that a group named **`kvm`** exists and that you are a member of that group. To take those into effect, run:

```
$ sudo groupadd kvm
$ sudo usermod -G $(groups | sed 's/ /,/g'),kvm $USER
$ newgrp kvm
```

The last command enrolls you to the kvm group without logging out of your session.

Networking

QEMU supports many ways of setting up networking for its guests, but for this post we'll use VDE.

You need to run several commands to prep the networking environment. Ideally, you'd want to save these in a shell function, or a shell script:

```
$ sudo vde_switch -tap tap0 -mod 660 -group kvm -daemon
$ sudo ip addr add 10.0.2.1/24 dev tap0
$ sudo ip link set dev tap0 up
$ sudo sysctl -w net.ipv4.ip_forward=1
$ sudo iptables -t nat -A POSTROUTING -s 10.0.2.0/24 -j MASQUERADE
```

The above commands will:

1. Create a VDE device
2. Configure the TCP/IP options for that device
3. Enable the VDE device
4. Enable packet forwarding on the host OS
5. Setup the routing configuration

Execution

Load the image

You now need to invoke **`qemu-kvm`**, the command that will launch everything up. The name of the command may differ with the one installed on your system.

If you're installing an OS from a bootable image, usually an ISO file, run:

```
$ sudo qemu-kvm -cpu host -m 2G -net nic,model=virtio \
-net vde -soundhw all -vga qxl \
-spice port=9999,addr=127.0.0.1,password=supersecretkey \
-boot once=d -cdrom installer.iso \
vm.qcow2
```

On subsequent uses:

```
$ sudo qemu-kvm -cpu host -m 2G -net nic,model=virtio \
-net vde -soundhw all -vga qxl \
-spice port=9999,addr=127.0.0.1,password=supersecretkey \
vm.qcow2
```

Let's break that down:

```
-cpu host
```

Use the KVM processor with all the supported features

```
-m 2G
```

Allocate 2GiB of host memory for the guest. Adjust as necessary.

```
-net nic,model=virtio -net vde
```

Create a virtual NIC, and enable VDE networking

```
-soundhw all
```

Enable all audio drivers

```
-vga qxl
```

Specify the video adapter to emulate. Use QXL when using SPICE

```
-spice addr=127.0.0.1,port=9999,password=supersecretkey
```

Specify the options for SPICE, separated by commas. *addr* and *port* are the IP address and TCP port that SPICE will listen on. Ideally, access to that port must be properly configured, and secured. *password* is key that will be used by the SPICE client, *spicec*, to connect to the guest display later.

```
-boot once=d -cdrom installer.iso
```

Boot initially from **`installer.iso`**, then on subsequent boots, boot in the normal order.

Running the *qemu-kvm* command above will load the image, but you won't be able to view the display, yet.

Connect to the SPICE display

To be able to use the guest machine's display, you need to connect to the SPICE server, using the SPICE client **`spicy`**:

```
$ spicy -h 127.0.0.1 -p 5901 -w supersecretkey
```

Take note that closing the spicy window will not kill the QEMU session. If the guest OS captures the mouse input, press `Shift+F12`, to get out of it.

Configure guest networking

Next, you need to properly configure the network configuration of the guest OS so that it can connect to the rest of the local network, and to the internet if the host machine has access to it.

IP address:

```
10.0.2.2
```

Default gateway:

```
10.0.2.1
```

DNS servers:

```
8.8.8.8
8.8.4.4
```

Closing the curtains

Restore networking

If you want to explicitly revert the network configuration, do the following.

```
$ sudo iptables -t nat -D POSTROUTING -s 10.0.2.0/24 \
-j MASQUERADE
$ sudo sysctl -w net.ipv4.ip_forward=0
$ sudo ip link set dev tap0 down
$ sudo ip link delete tap0
$ sudo pkill -9 vde_switch
$ sudo rm -f /run/vde.ctl/ctl
```

The above commands will:

1. Revert the routing configuration
2. Disable packet forwarding
3. Disable the VDE device
4. Delete the VDE device
5. Kill the VDE process
6. Remove control files

Putting it all

Here are all the commands from above, compiled into functions, so that they can be ran from the command line:

```
function kvm-net () {
  case $1 in
    up)
      sudo vde_switch -tap tap0 -mod 660 -group kvm -daemon
      sudo ip addr add 10.0.2.1/24 dev tap0
      sudo ip link set dev tap0 up
      sudo sysctl -w net.ipv4.ip_forward=1
      sudo iptables -t nat -A POSTROUTING -s 10.0.2.0/24 -j MASQUERA
      ;;
    down)
      sudo iptables -t nat -D POSTROUTING -s 10.0.2.0/24 -j MASQUERA
      sudo sysctl -w net.ipv4.ip_forward=0
      sudo ip link set dev tap0 down
      sudo ip link delete tap0
      sudo pkill -9 vde_switch
      sudo rm -f /run/vde.ctl/ctl
      ;;
  esac
}

function kvm-boot () {
    sudo qemu-kvm -cpu host -m 2G -net nic,model=virtio -net vde \
    -soundhw all -vga qxl \
    -spice port=6900,addr=127.0.0.1,disable-ticketing \
    $@
}

function kvm-iso () {
```

```
    kvm-boot -boot once=d -cdrom $1 ${argv[2,-1]}
}

function kvm-display () {
    spicy -p 6900 -h 127.0.0.1
}
```

I'll walk you through.

Initially, setup the netoworking:

```
$ kvm-net up
```

Then, load the image:

```
$ kvm-boot vm.qcow2
```

Finally, connect to the display:

```
$ kvm-display
```

When you're done with the VM, close the Spice display then shutdown the KVM networking:

```
$ kvm-net down
```

Closing remarks

QEMU supports a myriad of cool options that we've not even discussed here, including saving and loading states (snapshots), creating screen and audio grabs, and a whole lot more. To learn more about them, click here.

QEMU with KVM is a powerful, fast, and flexible solution for doing Full Virtualization. At least in my case, it out-performs the well-known options in the market. If you want to contribute to this project, head over to their GitHub page.

I hope this post helped you, in one way or another, learn more about QEMU and KVM and what it has to offer.
