By Falko Timme

Published: 2008-12-11 18:46

Virtualization With KVM On Ubuntu 8.10

Version 1.0
Author: Falko Timme <ft [at] falkotimme [dot] com>
Last edited 12/10/2008

This guide explains how you can install and use KVM for creating and running virtual machines on an Ubuntu 8.10 server. I will show how to create image-based virtual machines and also virtual machines that use a logical volume (LVM). KVM is short for **Kernel-based Virtual Machine** and makes use of hardware virtualization, i.e., you need a CPU that supports hardware virtualization, e.g. Intel VT or AMD-V.

I do not issue any guarantee that this will work for you!

1 Preliminary Note

I'm using a machine with the hostname *server1.example.com* and the IP address *192.168.0.100* here as my KVM host.

Because we will run all the steps from this tutorial with root privileges, we can either prepend all commands in this tutorial with the string *sudo*, or we become root right now by typing

```
sudo su
```

2 Installing KVM And vmbuilder

First check if your CPU supports hardware virtualization - if this is the case, the command

```
egrep '(vmx|svm)' --color=always /proc/cpuinfo
```

should display something, e.g. like this:

```
root@server1:~# egrep '(vmx|svm)' --color=always /proc/cpuinfo
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscal
l nx mmxext
 fxsr_opt rdtscp lm 3dnowext 3dnow rep_good nopl pni cx16 lahf_lm cmp_legacy svm extapic cr8_legacy 3dnowprefetch
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscal
l nx mmxext
 fxsr_opt rdtscp lm 3dnowext 3dnow rep_good nopl pni cx16 lahf_lm cmp_legacy svm extapic cr8_legacy 3dnowprefetch
root@server1:~#
```

If nothing is displayed, then your processor doesn't support hardware virtualization, and you must stop here.

To install KVM and `vmbuilder` (a script to create Ubuntu-based virtual machines), we run

```
apt-get install ubuntu-virt-server python-vm-builder
```

Afterwards we must add the user as which we're currently logged in (`root`) to the group `libvirtd`:

```
adduser `id -un` libvirtd
```

You need to log out and log back in for the new group membership to take effect.

To check if KVM has successfully been installed, run

```
virsh -c qemu:///system list
```

It should display something like this:

```
root@server1:~# virsh -c qemu:///system list
```

```
Connecting to uri: qemu:///system
 Id Name                 State
----------------------------------

root@server1:~#
```

If it displays an error instead, then something went wrong.

Next we need to set up a network bridge on our server so that our virtual machines can be accessed from other hosts as if they were physical systems in the network.

To do this, we install the package *bridge-utils*...

```
apt-get install bridge-utils
```

... and configure a bridge. Open */etc/network/interfaces*:

```
vi /etc/network/interfaces
```

Before the modification, my file looks as follows:

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
```

```
iface eth0 inet static
        address 192.168.0.100
        netmask 255.255.255.0
        network 192.168.0.0
        broadcast 192.168.0.255
        gateway 192.168.0.1
```

I change it so that it looks like this:

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet manual

auto br0
iface br0 inet static
        address 192.168.0.100
        network 192.168.0.0
        netmask 255.255.255.0
        broadcast 192.168.0.255
        gateway 192.168.0.1
        bridge_ports eth0
        bridge_fd 9
        bridge_hello 2
        bridge_maxage 12
        bridge_stp off
```

(Make sure you use the correct settings for your network!)

Restart the network...

```
/etc/init.d/networking restart
```

... and run

```
ifconfig
```

It should now show the network bridge (`br0`):

```
root@server1:~# ifconfig
br0       Link encap:Ethernet  HWaddr 00:1e:90:f3:f0:02
          inet addr:192.168.0.100  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::21e:90ff:fef3:f002/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:23 errors:0 dropped:0 overruns:0 frame:0
          TX packets:24 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1580 (1.5 KB)  TX bytes:2356 (2.3 KB)

eth0      Link encap:Ethernet  HWaddr 00:1e:90:f3:f0:02
          inet6 addr: fe80::21e:90ff:fef3:f002/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:13539 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7684 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:19476849 (19.4 MB)  TX bytes:647692 (647.6 KB)
          Interrupt:251 Base address:0xe000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

vnet0     Link encap:Ethernet  HWaddr 3e:7c:6f:ab:0e:8c
          inet addr:192.168.122.1  Bcast:192.168.122.255  Mask:255.255.255.0
          inet6 addr: fe80::3c7c:6fff:feab:e8c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:468 (468.0 B)

root@server1:~#
```

3 Creating An Image-Based VM

We can now create our first VM - an image-based VM (if you expect lots of traffic and many read- and write operations for that VM, use an LVM-based VM instead as shown in chapter 6 - image-based VMs are heavy on hard disk IO).

We will create a new directory for each VM that we want to create, e.g. `~/vm1`, `~/vm2`, `~/vm3`, and so on, because each VM will have a subdirectory called `ubuntu-kvm`, and obviously there can be just one such directory in `~/vm1`, for example. If you try to create a second VM in `~/vm1`, for example, you will get an error message saying `ubuntu-kvm already exists` (unless you run `vmbuilder` with the `--dest=DESTDIR` argument):

```
root@server1:~/vm1# vmbuilder kvm ubuntu -c vm2.cfg
```

```
2008-12-10 16:32:44,185 INFO     Cleaning up
ubuntu-kvm already exists
root@server1:~/vm1#
```

We will use the `vmbuilder` tool to create VMs. (You can learn more about `vmbuilder` **here**.) `vmbuilder` uses a template to create virtual machines - this template is located in the `/etc/vmbuilder/libvirt/` directory. Because we must modify the template, we create a copy and modify that one:

```
mkdir -p ~/vm1/mytemplates/libvirt

cp /etc/vmbuilder/libvirt/* ~/vm1/mytemplates/libvirt/
```

Now we open `~/vm1/mytemplates/libvirt/libvirtxml.tmpl`...

```
vi ~/vm1/mytemplates/libvirt/libvirtxml.tmpl
```

... and change the network section from

```
[...]
    <interface type='network'>
      <source network='default'/>
    </interface>
[...]
```

to

```
[...]
    <interface type='bridge'>
      <source bridge='br0'/>
```

```
   </interface>
[...]
```

because we want the VM to use our network bridge.

Now we come to the partitioning of our VM. We create a file called *vmbuilder.partition*...

```
vi ~/vm1/vmbuilder.partition
```

... and define the desired partitions as follows:

```
root 8000
swap 4000
---
/var 20000
```

This defines a root partition (/) with a size of 8000MB, a swap partition of 4000MB, and a */var* partition of 20000MB. The `---` line makes that the following partition (*/var* in this example) is on a separate disk image (i.e., this would create two disk images, one for root and swap and one for */var*). Of course, you are free to define whatever partitions you like (as long as you also define root and swap), and of course, they can be in just one disk image - this is just an example.

I want to install *openssh-server* in the VM. To make sure that each VM gets a unique OpenSSH key, we cannot install *openssh-server* when we create the VM. Therefore we create a script called *boot.sh* that will be executed when the VM is booted for the first time. It will install *openssh-server* (with a unique key) and also force the user (I will use the default username *administrator* for my VMs together with the default password *howtoforge*) to change the password when he logs in for the first time:

```
vi ~/vm1/boot.sh
```

```
# This script will run the first time the virtual machine boots
# It is ran as root.

# Expire the user account
passwd -e administrator

# Install openssh-server
apt-get update
apt-get install -qqy --force-yes openssh-server
```

Make sure you replace the username `administrator` with your default login name.

(You can find more about this here: **https://help.ubuntu.com/community/JeOSVMBuilder#First%20boot**)

(You can also define a "first login" script as described here: **https://help.ubuntu.com/community/JeOSVMBuilder#First%20login**)

Whenever `vmbuilder` builds a new VM, it has to download all packages from an Ubuntu mirror which can take quite some time. To speed this up, we install `apt-proxy`...

```
apt-get install apt-proxy
```

... to cache the downloaded packages so that subsequent VM installations will be a lot faster.

Now open `/etc/apt-proxy/apt-proxy-v2.conf`...

```
vi /etc/apt-proxy/apt-proxy-v2.conf
```

... and replace the default Ubuntu mirror with a mirror close to you (e.g. `http://de.archive.ubuntu.com/ubuntu` if you are in Germany):

```
[...]
[ubuntu]
;; Ubuntu archive
backends = http://de.archive.ubuntu.com/ubuntu
min_refresh_delay = 15m
[...]
```

Then we restart apt-proxy:

```
/etc/init.d/apt-proxy restart
```

apt-proxy listens on port *9999*, so we can pass our local apt-proxy "mirror" as an argument to the *vmbuilder* script.

Now take a look at

```
vmbuilder kvm ubuntu --help
```

to learn about the available options.

To create our first VM, *vm1*, we go to the VM directory...

```
cd ~/vm1/
```

... and run *vmbuilder*, e.g. as follows:

```
vmbuilder  kvm  ubuntu  --suite=intrepid  --flavour=virtual  --arch=amd64  --mirror=http://192.168.0.100:9999/ubuntu  -o  --libvirt=qemu:///system
--tmpfs=-  --ip=192.168.0.101  --part=vmbuilder.partition  --templates=mytemplates  --user=administrator  --name=Administrator  --pass=howtoforge
--addpkg=vim-nox --addpkg=unattended-upgrades --addpkg=acpid --firstboot=boot.sh --mem=256 --hostname=vm1
```

Most of the options are self-explanatory. `--part` specifies the file with the partitioning details, relative to our working directory (that's why we had to go to our VM directory before running `vmbuilder`), `--templates` specifies the directory that holds the template file (again relative to our working directory), and `--firstboot` specifies the firstboot script. `--libvirt=qemu:///system` tells KVM to add this VM to the list of available virtual machines. `--addpkg` allows you to specify Ubuntu packages that you want to have installed during the VM creation (see above why you shouldn't add `openssh-server` to that list and use the firstboot script instead).

In the `--mirror` line I have specified my local apt-proxy mirror (`http://192.168.0.100:9999/ubuntu`) - I have used my publically accessible IP address instead of `localhost` or `127.0.0.1` because this mirror will be used in the VM's `/etc/apt/sources.list` file as well, and of course, the VM won't be able to connect to `127.0.0.1` on the host. Of course, you can as well specify an official Ubuntu repository in `--mirror`, e.g. `http://de.archive.ubuntu.com/ubuntu`. If you leave out `--mirror`, then the default Ubuntu repository (`http://archive.ubuntu.com/ubuntu`) will be used.

The build process can take a few minutes.

Afterwards, you can find an XML configuration file for the VM in `/etc/libvirt/qemu/` (=> `/etc/libvirt/qemu/vm1.xml`):

```
ls -l /etc/libvirt/qemu/
```

```
root@server1:~/vm1# ls -l /etc/libvirt/qemu/
total 8
drwxr-xr-x 3 root root 4096 2008-12-10 15:26 networks
-rw------- 1 root root  963 2008-12-10 16:25 vm1.xml
root@server1:~/vm1#
```

The disk images are located in the `ubuntu-kvm/` subdirectory of our VM directory:

```
ls -l ~/vm1/ubuntu-kvm/
```

```
root@server1:~/vm1# ls -l ~/vm1/ubuntu-kvm/
total 402804
-rw-r--r-- 1 root root 240963584 2008-12-10 16:37 disk0.qcow2
```

```
-rw-r--r-- 1 root root 171094016 2008-12-10 16:37 disk1.qcow2
root@server1:~/vm1#
```

4 Creating A Second VM

If you want to create a second VM (`vm2`), here's a short summary of the commands:

```
mkdir -p ~/vm2/mytemplates/libvirt

cp /etc/vmbuilder/libvirt/* ~/vm2/mytemplates/libvirt/
```

```
vi ~/vm2/mytemplates/libvirt/libvirtxml.tmpl
```

```
vi ~/vm2/vmbuilder.partition
```

```
vi ~/vm2/boot.sh
```

```
cd ~/vm2/

vmbuilder  kvm  ubuntu  --suite=intrepid  --flavour=virtual  --arch=amd64  --mirror=http://192.168.0.100:9999/ubuntu  -o  --libvirt=qemu:///system
--tmpfs=-  --ip=192.168.0.102  --part=vmbuilder.partition  --templates=mytemplates  --user=administrator  --name=Administrator  --pass=howtoforge
--addpkg=vim-nox --addpkg=unattended-upgrades --addpkg=acpid --firstboot=boot.sh --mem=256 --hostname=vm2
```

(Please note that you don't have to create a new directory for the VM (`~/vm2`) if you pass the `--dest=DESTDIR` argument to the `vmbuilder` command - it allows you to create a VM in a directory where you've already created another VM. In that case you don't have to create new `vmbuilder.partition` and `boot.sh` files and don't have to modify the template, but can simply use the existing files:

```
cd ~/vm1/
```

```
vmbuilder kvm ubuntu --suite=intrepid --flavour=virtual --arch=amd64 --mirror=http://192.168.0.100:9999/ubuntu -o --libvirt=qemu:///system --tmpfs=-
--ip=192.168.0.102 --part=vmbuilder.partition --templates=mytemplates --user=administrator --name=Administrator --pass=howtoforge --addpkg=vim-nox
--addpkg=unattended-upgrades --addpkg=acpid --firstboot=boot.sh --mem=256 --hostname=vm2 --destdir=vm2-kvm
```

)

5 Managing A VM

VMs can be managed through `virsh`, the "virtual shell". To connect to the virtual shell, run

```
virsh --connect qemu:///system
```

This is how the virtual shell looks:

```
root@server1:~/vm1/ubuntu-kvm# virsh --connect qemu:///system
Connecting to uri: qemu:///system
Welcome to virsh, the virtualization interactive terminal.

Type:  'help' for help with commands
       'quit' to quit

virsh #
```

You can now type in commands on the virtual shell to manage your VMs. Run

```
help
```

to get a list of available commands:

```
virsh # help
Commands:

    help            print help
    attach-device   attach device from an XML file
    attach-disk     attach disk device
    attach-interface attach network interface
    autostart       autostart a domain
    capabilities    capabilities
    connect         (re)connect to hypervisor
    console         connect to the guest console
    create          create a domain from an XML file
    start           start a (previously defined) inactive domain
    destroy         destroy a domain
    detach-device   detach device from an XML file
    detach-disk     detach disk device
    detach-interface detach network interface
    define          define (but don't start) a domain from an XML file
    domid           convert a domain name or UUID to domain id
    domuuid         convert a domain name or id to domain UUID
    dominfo         domain information
    domname         convert a domain id or UUID to domain name
    domstate        domain state
    domblkstat      get device block stats for a domain
    domifstat       get network interface stats for a domain
    dumpxml         domain information in XML
    freecell        NUMA free memory
    hostname        print the hypervisor hostname
    list            list domains
    migrate         migrate domain to another host
    net-autostart   autostart a network
    net-create      create a network from an XML file
    net-define      define (but don't start) a network from an XML file
    net-destroy     destroy a network
    net-dumpxml     network information in XML
    net-list        list networks
    net-name        convert a network UUID to network name
    net-start       start a (previously defined) inactive network
    net-undefine    undefine an inactive network
    net-uuid        convert a network name to network UUID
    nodeinfo        node information
    pool-autostart  autostart a pool
    pool-build      build a pool
    pool-create     create a pool from an XML file
    pool-create-as  create a pool from a set of args
    pool-define     define (but don't start) a pool from an XML file
    pool-define-as  define a pool from a set of args
    pool-destroy    destroy a pool
    pool-delete     delete a pool
    pool-dumpxml    pool information in XML
    pool-info       storage pool information
    pool-list       list pools
    pool-name       convert a pool UUID to pool name
    pool-refresh    refresh a pool
    pool-start      start a (previously defined) inactive pool
    pool-undefine   undefine an inactive pool
    pool-uuid       convert a pool name to pool UUID
    quit            quit this interactive terminal
    reboot          reboot a domain
    restore         restore a domain from a saved state in a file
    resume          resume a domain
    save            save a domain state to a file
    schedinfo       show/set scheduler parameters
    dump            dump the core of a domain to a file for analysis
    shutdown        gracefully shutdown a domain
    setmem          change memory allocation
    setmaxmem       change maximum memory limit
    setvcpus        change number of virtual CPUs
    suspend         suspend a domain
    ttyconsole      tty console
    undefine        undefine an inactive domain
    uri             print the hypervisor canonical URI
    vol-create      create a vol from an XML file
    vol-create-as   create a volume from a set of args
    vol-delete      delete a vol
    vol-dumpxml     vol information in XML
    vol-info        storage vol information
    vol-list        list vols
    vol-path        convert a vol UUID to vol path
    vol-name        convert a vol UUID to vol name
    vol-key         convert a vol UUID to vol key
    vcpuinfo        domain vcpu information
    vcpupin         control domain vcpu affinity
    version         show version
    vncdisplay      vnc display

virsh #
```

```
list
```

shows all running VMs;

```
list --all
```

shows all VMs, running and inactive:

```
virsh # list --all
```

```
 Id Name                 State
----------------------------------
  - vm1                  shut off

virsh #
```

Before you start a new VM for the first time, you must define it from its xml file (located in the */etc/libvirt/qemu/* directory):

```
define /etc/libvirt/qemu/vm1.xml
```

Please note that whenever you modify the VM's xml file in */etc/libvirt/qemu/*, you must run the *define* command again!

Now you can start the VM:

```
start vm1
```

After a few moments, you should be able to connect to the VM with an SSH client such as **PuTTY**; log in with the default username and password. After the first login you will be prompted to change the password.

```
list
```

should now show the VM as running:

```
virsh # list
 Id Name                 State
----------------------------------
  1 vm1                  running

virsh #
```

To stop a VM, run

```
shutdown vm1
```

To immediately stop it (i.e., pull the power plug), run

```
destroy vm1
```

Suspend a VM:

```
suspend vm1
```

Resume a VM:

```
resume vm1
```

These are the most important commands.

Type

```
quit
```

to leave the virtual shell.

6 Creating An LVM-Based VM

LVM-based VMs have some advantages over image-based VMs. They are not as heavy on hard disk IO, and they are easier to back up (using **LVM snapshots**).

To use LVM-based VMs, you need a volume group that has some free space that is not allocated to any logical volume. In this example, I use the volume group `/dev/vg01` with a size of approx. 454GB...

```
vgdisplay
```

```
root@server1:~# vgdisplay
  --- Volume group ---
  VG Name               vg01
  System ID
  Format                lvm2
  Metadata Areas        1
  Metadata Sequence No  2
  VG Access             read/write
  VG Status             resizable
  MAX LV                0
  Cur LV                1
  Open LV               1
  Max PV                0
  Cur PV                1
  Act PV                1
  VG Size               454.67 GB
  PE Size               4.00 MB
  Total PE              116396
  Alloc PE / Size       75000 / 292.97 GB
  Free  PE / Size       41396 / 161.70 GB
  VG UUID               q3xIiX-LDlm-IbMu-2PK2-WVoc-zHb8-8ibb32

root@server1:~#
```

... that contains the logical volume `/dev/vg01/root` with a size of approx. 292GB - the rest is not allocated and can be used for VMs:

```
lvdisplay
```

```
root@server1:~# lvdisplay
  --- Logical volume ---
  LV Name                /dev/vg01/root
  VG Name                vg01
  LV UUID                f9W43z-RC1i-9JE8-CvOS-Qa89-0STq-q1M71e
  LV Write Access        read/write
  LV Status              available
  # open                 1
  LV Size                292.97 GB
  Current LE             75000
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           254:0

root@server1:~#
```

I will now create the virtual machine *vm5* as an LVM-based VM. We can use the *vmbuilder* command again. *vmbuilder* knows the *--raw* option which allows to write the VM to a block device (e.g. */dev/vg01/vm5*) - I've tried this, and it gave back no errors, however, I was not able to boot the VM (*start vm5* didn't show any errors either, but I've never been able to access the VM). Therefore, I will create *vm5* as an image-based VM first and then convert it into an LVM-based VM.

```
mkdir -p ~/vm5/mytemplates/libvirt

cp /etc/vmbuilder/libvirt/* ~/vm5/mytemplates/libvirt/
```

```
vi ~/vm5/mytemplates/libvirt/libvirtxml.tmpl
```

Make sure that you create all partitions in just one image file, so don't use `---` in the *vmbuilder.partition* file:

```
vi ~/vm5/vmbuilder.partition
```

```
root 8000
swap 2000
/var 10000
```

```
vi ~/vm5/boot.sh
```

```
cd ~/vm5/

vmbuilder kvm ubuntu --suite=intrepid --flavour=virtual --arch=amd64 --mirror=http://192.168.0.100:9999/ubuntu -o --libvirt=qemu:///system
--tmpfs=- --ip=192.168.0.105 --part=vmbuilder.partition --templates=mytemplates --user=administrator --name=Administrator --pass=howtoforge
--addpkg=vim-nox --addpkg=unattended-upgrades --addpkg=acpid --firstboot=boot.sh --mem=256 --hostname=vm5
```

As you see from the *vmbuilder.partition* file, the VM will use a max. of 20GB, so we create a logical volume called */dev/vg01/vm5* with a size of 20GB now:

```
lvcreate -L20G -n vm5 vg01
```

Don't create a file system in the new logical volume!

We will use the *qemu-img* command to convert the image to an LVM-based VM. The *qemu-img* command is part of the *qemu* package which we must install now:

```
apt-get install qemu
```

Then we go to the VM's *ubuntu-kvm/* directory...

```
cd ~/vm5/ubuntu-kvm/
```

... and convert the image as follows:

```
qemu-img convert disk0.qcow2 -O raw /dev/vg01/vm5
```

Afterwards you can delete the disk image:

```
rm -f disk0.qcow2
```

Now we must open the VM's xml configuration file */etc/libvirt/qemu/vm5.xml*...

```
vi /etc/libvirt/qemu/vm5.xml
```

... and change the following section...

```
[...]
    <disk type='file' device='disk'>
      <source file='/root/vm5/ubuntu-kvm/disk0.qcow2'/>
      <target dev='hda' bus='ide'/>
    </disk>
[...]
```

... so that it looks as follows:

```
[...]
    <disk type='file' device='disk'>
      <source file='/dev/vg01/vm5'/>
      <target dev='hda' bus='ide'/>
    </disk>
[...]
```

That's it! You can now use `virsh` to manage the VM.

7 Links

- KVM (Ubuntu Community Documentation): **https://help.ubuntu.com/community/KVM**
- vmbuilder: **https://help.ubuntu.com/community/JeOSVMBuilder**
- JeOS and vmbuilder: **http://doc.ubuntu.com/ubuntu/serverguide/C/jeos-and-vmbuilder.html**
- Ubuntu: **http://www.ubuntu.com/**