BSc in Computer Engineering
CMP4103
Computer Systems and Network Security

Lecture 5(b)

KVM Hipervisor lab

Eng Diarmuid O'Briain, CEng, CISSP

Department of Electrical and Computer Engineering,
College of Engineering, Design, Art and Technology,
Makerere University

Author

Diarmuid is a Chartered Engineer (CEng) with over 25 years experience in Telecommunications, Information Networking and Security. He has designed and implemented next-generation networks and information security solutions for major multi-national communications companies as well as serving as Chief Technical Officer for an Irish Internet Service Provider for over 5 years. Since 1999 he has also lectured on Telecommunications and Computing programmes at both the Dublin Institute of Technology (DIT), the Institute of Technology, Carlow (ITC) in Ireland and the College of Engineering, Design, Art and Technology (CEDAT) at Makerere University in Uganda.

Table of Contents

Illustration Index

1. Terminology

- **node** is a single physical machine.
- **hypervisor** is a layer of software allowing to virtualise a node in a set of Virtual Machines (VM) with possibly different configurations than the node itself.
- **KVM** is a Kernel-based Virtual Machine (KVM) is the GNU/Linux Hypervisor.
- **domain** is an instance of an Operating System (OS) (or subsystem in the case of container virtualisation) running on a virtualised machine provided by the hypervisor.
- **QEMU** is a Quick EMUlator (QEMU), a generic and open source machine emulator and virtualiser for I/O hardware emulation.
- **libvirt** is a toolkit virtualisation management system to interact with the virtualisation capabilities of recent versions of Linux, KVM, Xen and LXC.
- **libvirtd** is the server side daemon component of the libvirt virtualisation management system.
- **virsh** is a command line interface tool for managing guests and the hypervisor.
- **virt-manager** is a desktop user interface for managing virtual machines through libvirt.
- **virt-viewer** is a lightweight interface for interacting with the graphical display of virtualised guest OS. It can display VNC or SPICE, and uses libvirt to lookup the graphical connection details.
- **virt-clone** is a command line tool for cloning existing inactive guests. It copies the disk images, and defines a configuration with new name, UUID and MAC address pointing to the copied disks.
- **spicec** is a Simple Protocol for Independent Computing Environments (SPICE) graphical client. SPICE is said to be faster than VNC as a remote desktop protocol.
- **vncviewer** is a Virtual Network Computing (VNC) graphical client.

2. Pre-installation check

2.1 Enable virtualisation support in BIOS

To support HVM guests, virtualisation extensions need to be enabled in the BIOS. In the BIOS the Virtualise option appears under “Advanced Chipset Features”as one of the following:

- Enable Virtualisation Technology.
- Enable Intel VT.
- Vanderpool Technology.

Confirm that the hardware virtualisation is now supported by the CPU by searching for *vmx* to see if the computer has an Intel processor or *svm* for AMD support if the hardware has an AMD processor.

Check that the CPU supports hardware virtualisation. 0 means that the CPU doesn't support hardware virtualisation while > 0 means it does but it still needs to be enabled in the BIOS. (ie. *vmx* or *svm* has appeared x number of times in the output of the command.

```
$ egrep -c '(vmx|svm)' /proc/cpuinfo
4
```

Check if a 64 bit kernel is running. 0 means that the CPU is not 64-bit. *lm* stands for Long Mode which equates to a 64-bit CPU.

```
$ egrep -c ' lm ' /proc/cpuinfo
8

$ uname -m
x86_64
```

3. Installation

KVM requires a number of elements to operate.

3.1 Install KVM packages

- **libvirt** is a C toolkit to interact with the virtualisation capabilities of GNU/Linux. The library provides a C API for different virtualisation mechanisms and currently supports QEMU, KVM, XEN, OpenVZ, LXC, and VirtualBox.
- **qemu-kvm** permits the running of multiple virtual computers, each running unmodified GNU/Linux or Windows images on X86 hardware. Each virtual machine has private virtualised hardware: a network card, disk, graphics adapter, etc.
- **virt-manager** - graphical user interface

```
$ sudo apt-get install qemu-kvm libvirt-bin virt-manager
```

3.2 GNU/Linux Bridge utilities

Without a bridge KVM VMs will only have network access to other VMs on the same server and to the host itself via a shared private network 192.168.122.0. To allow VMs access to the LAN, create a network bridge on the host.

```
$ sudo apt-get install bridge-utils
```

4. Setup Ethernet interface

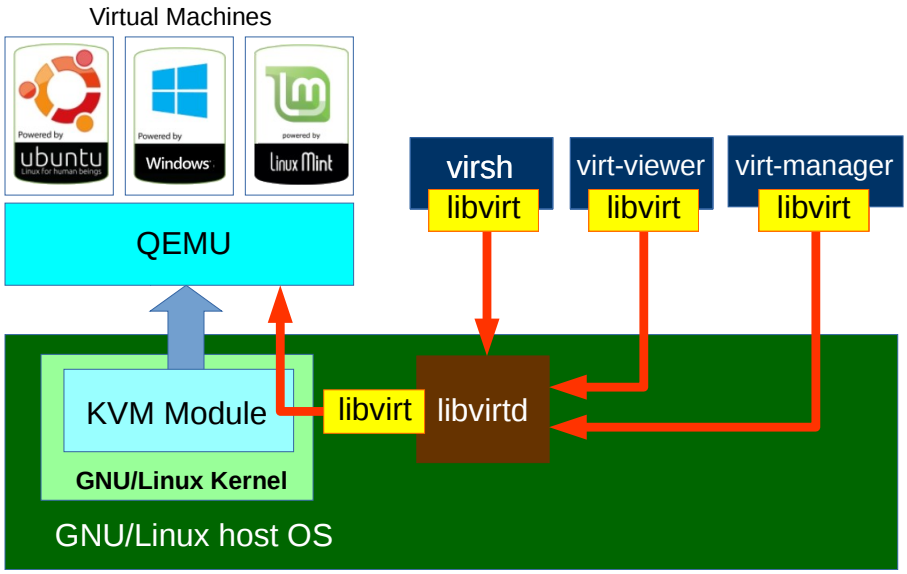

Illustration 1: KVM virtualisation block diagram

Edit the */etc/network/interfaces* file by creating a bridge, put *eth0* into the bridge and transfer the IP addressing information to the bridge. Restart networking.

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

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary interface eth0
auto eth0
iface eth0 inet manual

auto br0
iface br0 inet static
        address 213.74.34.100
        netmask 255.255.255.0
        network 213.74.34.0
        broadcast 213.74.34.255
        gateway 213.74.34.1
        dns-nameservers 8.8.8.8
        bridge_ports eth0
        bridge_fd 9
        bridge_hello 2
```

```
        bridge_maxage 12
        bridge_stp off

:wq!
```

Here is an explanation on some of the bridge settings.

- **bridge_ports eth0** - Define ports within the bridge.
- **bridge_fd 9** - Set bridge forward delay to 9 seconds (Default: 15).
- **bridge_maxage 12** - Set max message age to 12 seconds (Default: 20).
- **bridge_stp off** - Disables Spanning Tree Protocol.

Restart the *networking* service.

```
$ sudo /etc/init.d/networking restart
[ ok ] Restarting networking (via systemctl): networking.service.
```

Now confirm the IP settings.

```
$ ip addr list
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
pfifo_fast master br0 state UP group default qlen 1000
    link/ether 30:5a:3a:08:39:21 brd ff:ff:ff:ff:ff:ff
    inet 213.74.34.100/24 brd 213.74.34.255 scope global eth0
       valid_lft forever preferred_lft forever
3: br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
state UP group default
    link/ether 30:5a:3a:08:39:21 brd ff:ff:ff:ff:ff:ff
    inet 213.74.34.100/24 brd 213.74.34.255 scope global br0
       valid_lft forever preferred_lft forever
    inet6 fe80::325a:3aff:fe08:3921/64 scope link
       valid_lft forever preferred_lft forever
```

5. Add Users to Groups

Add the user to the *libvirtd* and *kvm* groups. Then logout and back in to activate.

```
$ sudo adduser `id -un` libvirt
Adding user 'alovelace' to group 'libvirt' ...

$ sudo adduser `id -un` kvm
Adding user 'alovelace' to group 'kvm' ...
```

Confirm entries.

```
$ sudo egrep '(kvm|libvirt)' /etc/group
libvirt:x:124:alovelace
kvm:x:125:alovelace
libvirt-qemu:x:126:libvirt-qemu
```

Logout of shell and log back in to join the groups. Use the *id* command to verify.

```
$ id
uid=1000(alovelace) gid=1000(alovelace)
groups=1000(alovelace),24(cdrom),25(floppy),27(sudo),29(audio),30(dip
),44(video),46(plugdev),108(netdev),111(scanner),115(bluetooth),124(l
ibvirt),125(kvm)
```

6. Verify installation

If the installation is correct then a list like this will be presented. As no VMs are generated as yet this list will be empty.

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

virsh # list
 Id    Name                           State
----------------------------------------------------
```

7. Display node information

To display node information for the hypervisor, the host machine that supports the virtualisation process use the *nodeinfo* command.

```
virsh # nodeinfo
CPU model:           x86_64
CPU(s):              4
CPU frequency:       800 MHz
CPU socket(s):       1
Core(s) per socket:  4
Thread(s) per core:  1
NUMA cell(s):        1
Memory size:         16151212 KiB
```

8. Creation of domains

The tools that will be used can be seen by typing *virt* and hitting <TAB>.

```
$ virt
virt-clone  virt-host-validate  virt-login-shell  virt-viewer
virt-convert  virt-image  virt-manager  virt-xml
virtfs-proxy-helper  virt-install  virt-pki-validate
virt-xml-validate
```

8.1 Domain directory

By default images are stored in the directory /var/lib/libvirt/images/ which was created automatically when *qemu-kvm* was installed. However the machine in use has a separate harddrive */dev/sdb/* for VMs and it is mounted as */virt*, therefore images will be stored in */virt/kvm/images/*.

8.2 OS variant

It is useful to optimise the guest configuration for the specific OS to be installed. Get a list of OS variants as follows.

```
$ virt-install --os-variant list
win2k8               : Microsoft Windows Server 2008 (or later)
win2k3               : Microsoft Windows Server 2003
win7                 : Microsoft Windows 7 (or later)
vista                : Microsoft Windows Vista
winxp64              : Microsoft Windows XP (x86_64)
winxp                : Microsoft Windows XP
win2k                : Microsoft Windows 2000
openbsd4             : OpenBSD 4.x (or later)
freebsd9             : FreeBSD 9.x
freebsd8             : FreeBSD 8.x
freebsd7             : FreeBSD 7.x
freebsd6             : FreeBSD 6.x
freebsd10            : FreeBSD 10.x (or later)
solaris9             : Sun Solaris 9
solaris11            : Sun Solaris 11 (or later)
solaris10            : Sun Solaris 10
opensolaris          : Sun OpenSolaris (or later)
netware6             : Novell Netware 6 (or later)
netware5             : Novell Netware 5
netware4             : Novell Netware 4
msdos                : MS-DOS
generic              : Generic
altlinux             : ALT Linux (or later)
debianwheezy         : Debian Wheezy (or later)
debiansqueeze        : Debian Squeeze
debianlenny          : Debian Lenny
debianetch           : Debian Etch
fedora20             : Fedora 20 (or later)
fedora19             : Fedora 19
fedora18             : Fedora 18
fedora17             : Fedora 17
fedora16             : Fedora 16
fedora15             : Fedora 15
fedora14             : Fedora 14
fedora13             : Fedora 13
fedora12             : Fedora 12
fedora11             : Fedora 11
fedora10             : Fedora 10
fedora9              : Fedora 9
fedora8              : Fedora 8
fedora7              : Fedora 7
fedora6              : Fedora Core 6
fedora5              : Fedora Core 5
mes5.1               : Mandriva Enterprise Server 5.1 (or later)
mes5                 : Mandriva Enterprise Server 5.0
mandriva2010         : Mandriva Linux 2010 (or later)
mandriva2009         : Mandriva Linux 2009 and earlier
```

```
mageia1              : Mageia 1 (or later)
rhel7                : Red Hat Enterprise Linux 7 (or later)
rhel6                : Red Hat Enterprise Linux 6
rhel5.4              : Red Hat Enterprise Linux 5.4 or later
rhel5                : Red Hat Enterprise Linux 5
rhel4                : Red Hat Enterprise Linux 4
rhel3                : Red Hat Enterprise Linux 3
rhel2.1              : Red Hat Enterprise Linux 2.1
sles11               : Suse Linux Enterprise Server 11 (or later)
sles10               : Suse Linux Enterprise Server
opensuse12           : openSuse 12 (or later)
opensuse11           : openSuse 11
ubuntusaucy          : Ubuntu 13.10 (Saucy Salamander) (or later)
ubunturaring         : Ubuntu 13.04 (Raring Ringtail)
ubuntuquantal        : Ubuntu 12.10 (Quantal Quetzal)
ubuntuprecise        : Ubuntu 12.04 LTS (Precise Pangolin)
ubuntuoneiric        : Ubuntu 11.10 (Oneiric Ocelot)
ubuntunatty          : Ubuntu 11.04 (Natty Narwhal)
ubuntumaverick       : Ubuntu 10.10 (Maverick Meerkat)
ubuntulucid          : Ubuntu 10.04 LTS (Lucid Lynx)
ubuntukarmic         : Ubuntu 9.10 (Karmic Koala)
ubuntujaunty         : Ubuntu 9.04 (Jaunty Jackalope)
ubuntuintrepid       : Ubuntu 8.10 (Intrepid Ibex)
ubuntuhardy          : Ubuntu 8.04 LTS (Hardy Heron)
mbs1                 : Mandriva Business Server 1 (or later)
virtio26             : Generic 2.6.25 or later kernel with virtio
generic26            : Generic 2.6.x kernel
generic24            : Generic 2.4.x kernel
```

8.3 Build first domain

As an example build a Debian 8.2 64-bit domain. This is achieved using *virt-install* which is a tool for creating new KVM, Xen, or GNU/Linux container guests using the "libvirt" hypervisor management library.

```
$ virt-install \
        --connect qemu:///system \
        --virt-type=kvm \
        --name vm01 \
        --ram 512 \
        --vcpus=2 \
        --disk path=/virt/kvm/images/vm01.img,size=12 \
        --cdrom /virt/iso/Debian-8.2-ISO/debian-8.2.0-amd64-CD.iso \
        --graphics vnc,listen=0.0.0.0 \
        --noautoconsole \
        --os-type linux \
        --os-variant debianwheezy \
        --network=bridge:br0 \
        --hvm
```

```
Starting install...
Allocating 'vm01.img'                     |  12 GB     00:00
Creating domain...                        |    0 B     00:00
Domain installation still in progress. You can reconnect to
the console to complete the installation process.
```

- **connect** - Connect to the hypervisor.
    - **qemu:**///**system** - For creating KVM and QEMU guests to be run by the system libvirtd instance.
- **virt-type** - The hypervisor to install on. Example choices are kvm, qemu, xen, or kqemu.
- **name** - Name of the new guest virtual machine instance.
- **ram** - RAM allocated for the guest, in megabytes.
- **disk** - Specifies media to use as storage for the guest.
    - **size** - Size (in GB) to use if creating new storage.
- **cdrom** - Source of installation, an ISO file.
- **graphics** - Setup either a VNC or Spice Server in the host that allows access.
    - **listen** - Defines either 127.0.0.1 (local access), 0.0.0.0 (global access or a specific IP address for access.
- **noautoconsole** - Don't automatically try to connect to the guest console.
- **os-type** - Optimize the guest configuration for a type of operating system (ex. *linux*, *windows*)
- **os-variant** - Further optimise the guest configuration for a specific OS (if this is used os-type is not strictly necessary).
- **network** - Connect the guest to the host network.
- **hvm** - Request the use of full virtualisation, if both para & full virtualisation are available on the host.

Note there is an option to use either the SPICE or VNC remote desktop protocol. The *virt-viewer* will connect to whichever is configured in the domain at build time. There is also an option to change a domain with the *virt-manager* domain details which will be see later. Essentially the SPICE protocol is generally faster than VNC.

8.4 Connect to a domain

To connect to the domain use the *virt-viewer* tool and either identify the VM by its *Id*, in this case *1* or by its *name* which in this case is *vm01*.

```
$ virt-viewer --connect qemu:///system 1
```

or

```
$ virt-viewer --connect qemu:///system vm01
```

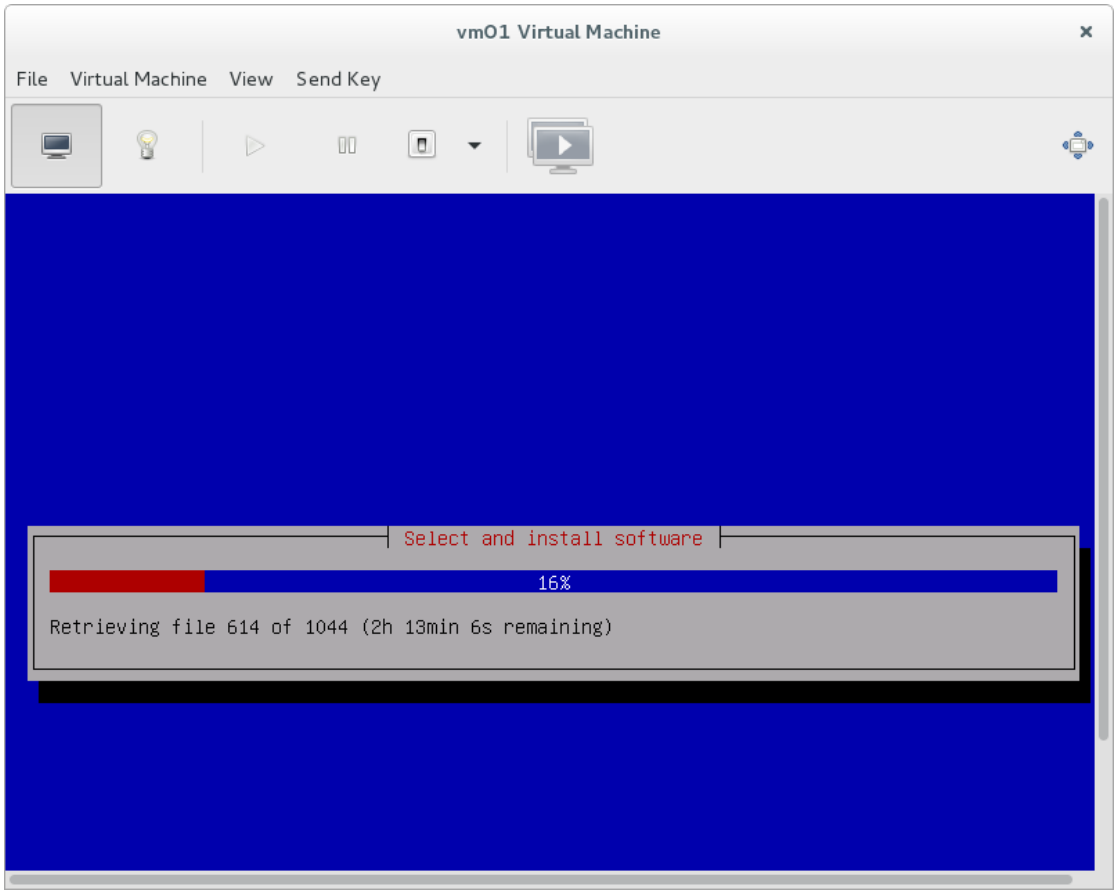

Illustration 2: Debian install

Follow the Debian install script and the domain guest will install.

Illustration 3: Debian guest

8.5 Build a Microsoft Windows domain

As a second example build a Microsoft Windows 7, 64-bit domain.

```
$ virt-install \
      --connect qemu:///system \
      --virt-type=kvm \
      --name vm02 \
      --ram 1024 \
      --vcpus=2 \
      --disk path=/virt/kvm/images/vm02.img,size=12 \
      --cdrom /virt/iso/Windows-7/Microsoft_Windows_7.iso \
      --graphics spice,listen=0.0.0.0 \
      --noautoconsole \
      --os-type windows \
      --os-variant win7 \
      --network=bridge:br0 \
      --hvm
```

```
Starting install...
Allocating 'vm02.img'                          |  12 GB     00:00
Creating domain...                             |    0 B     00:00
Domain installation still in progress. Waiting  for installation to
complete.
```

Connect to the running installation graphic.

```
$ virt-viewer --connect qemu:///system vm02
```

Illustration 4: Windows guest install

Follow the Microsoft Windows install process, it will reboot a number of times as part of the install which requires the *virt-viewer* to be restarted to reconnect.

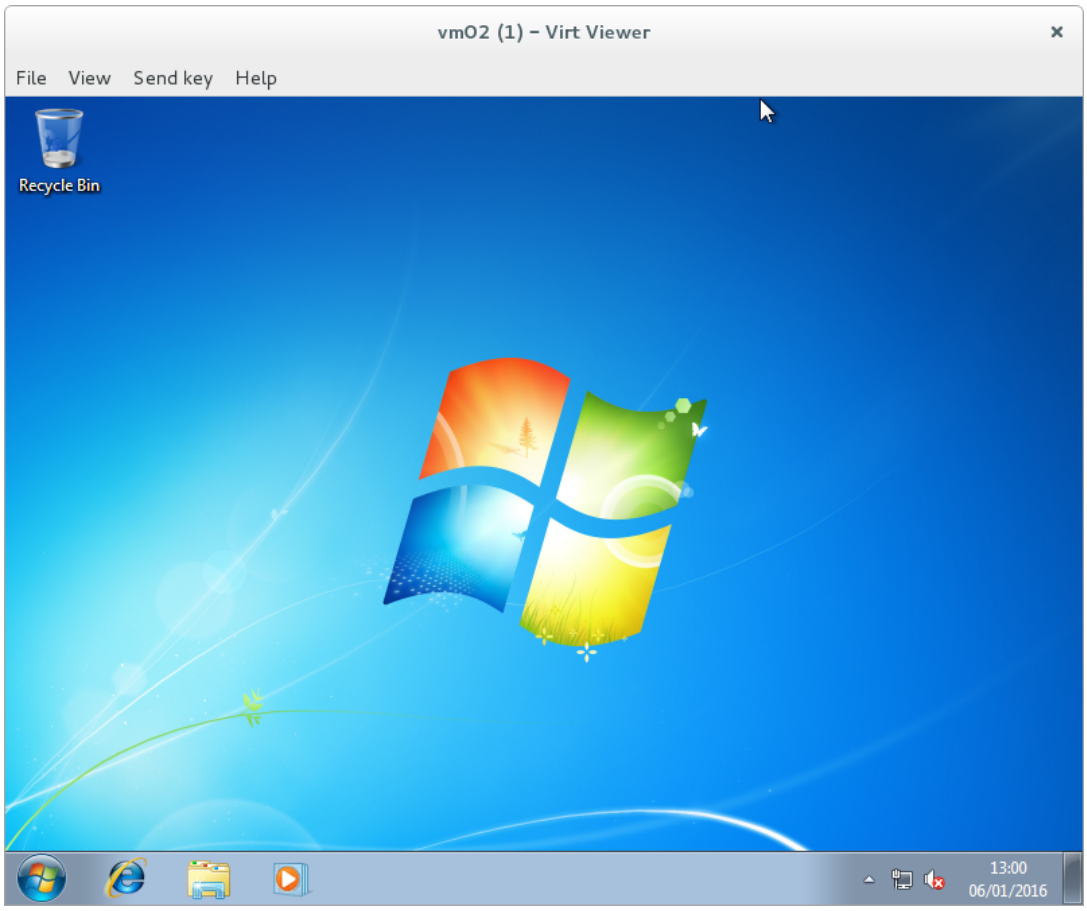

Illustration 5: Windows guest

8.6 Build a domain over the Internet

As an example build a Ubuntu 64-bit domain. This is achieved using *virt-install* which is a tool for creating new KVM, Xen, or GNU/Linux container guests using the *libvirt* hypervisor management library.

```
$ virt-install \
        --connect qemu:///system \
        --virt-type=kvm \
        --name vm03 \
        --ram 512 \
        --vcpus=2 \
        --disk path=/virt/kvm/images/vm03.img,size=12 \
        --location
'http://archive.ubuntu.com/ubuntu/dists/wily/main/installer-amd64/' \
        --graphics spice,listen=0.0.0.0 \
        --noautoconsole \
        --os-type linux \
        --os-variant ubuntusaucy \
        --network=bridge:br0 \
        --hvm
```

```
  Starting install...
  Retrieving file MANIFEST...
  Retrieving file linux...                21% [====================-
  Retrieving file linux...                21% [====================
  Retrieving file linux...
|  13 MB     00:55 ...
  Retrieving file initrd.gz...
|  44 MB     03:12 ...
  Allocating 'virtinst-linux.vj1SiL'
| 6.5 MB     00:00
  Transferring virtinst-linux.vj1SiL
| 6.5 MB     00:00
  Allocating 'virtinst-initrd.gz.FkqiPN'
|  22 MB     00:00
  Transferring virtinst-initrd.gz.FkqiPN
|  22 MB     00:00
  Allocating 'vm03.img'
|  12 GB     00:00
  Creating domain...
|    0 B     00:00
  Domain installation still in progress. You can reconnect to
  the console to complete the installation process.
```

- **location** - Used instead of *cdrom* when point to an image on the Internet. Note that the link specified is to the root directory and not to the *.iso* image itself.

8.7 Connect to a remote domain

Install a SPICE or VNC client.

8.7.1 SPICE

```
$ sudo apt-get install spice-client
$ sudo apt-get install spice-client-gtk
$ sudo apt-get install python-spice-client-gtk
```

Find the domain port number on the hypervisor and confirm it is configured for SPICE.

```
virsh # domdisplay vm05
spice://localhost:5905
```

Connect with the client.

```
$ spicec --host 213.74.34.100 --port 5905
```

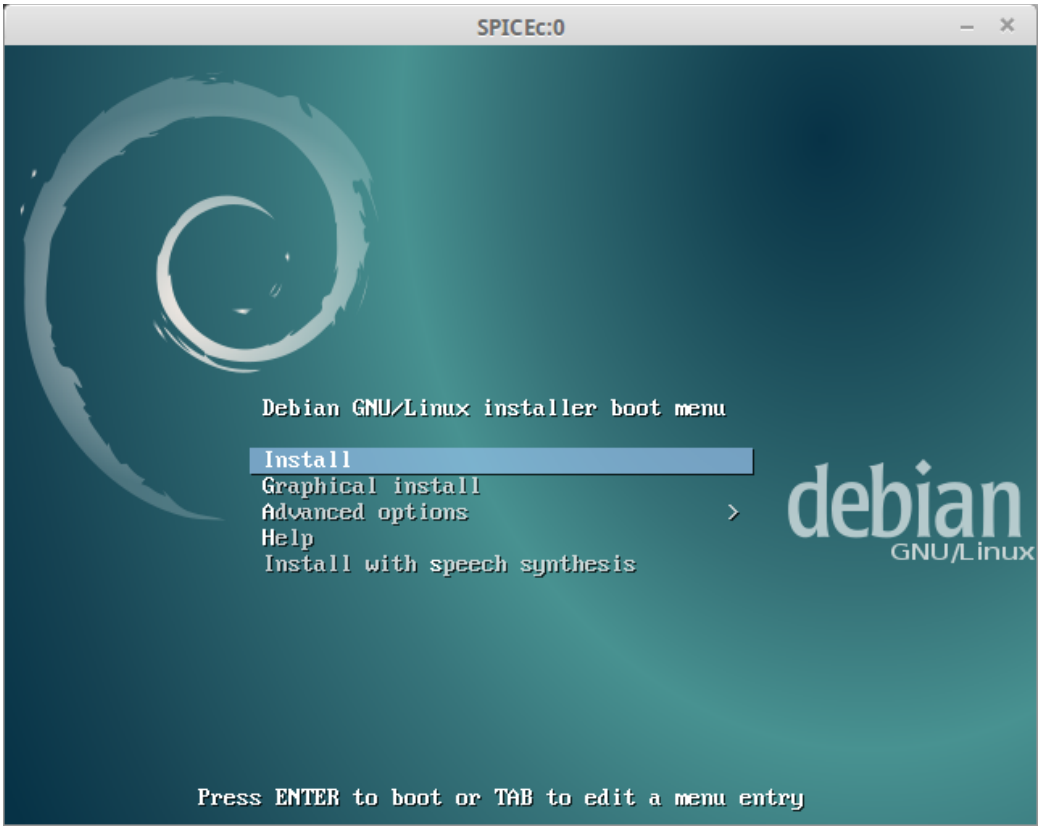

Illustration 6: Debian guest through SPICE viewer

For Microsoft Windows domains it will be necessary to install Windows guest tools. This can be downloaded from http://www.spice-space.org/download.html.

8.7.2 VNC

Install a VNC client.

```
$ sudo apt-get install vncviewer
```

Find the domain port number on the hypervisor and confirm it is configured for VNC.

```
virsh # domdisplay vm02
vnc://localhost:8
```

Connect with the client

```
$ vncviewer 213.74.34.100:8
```

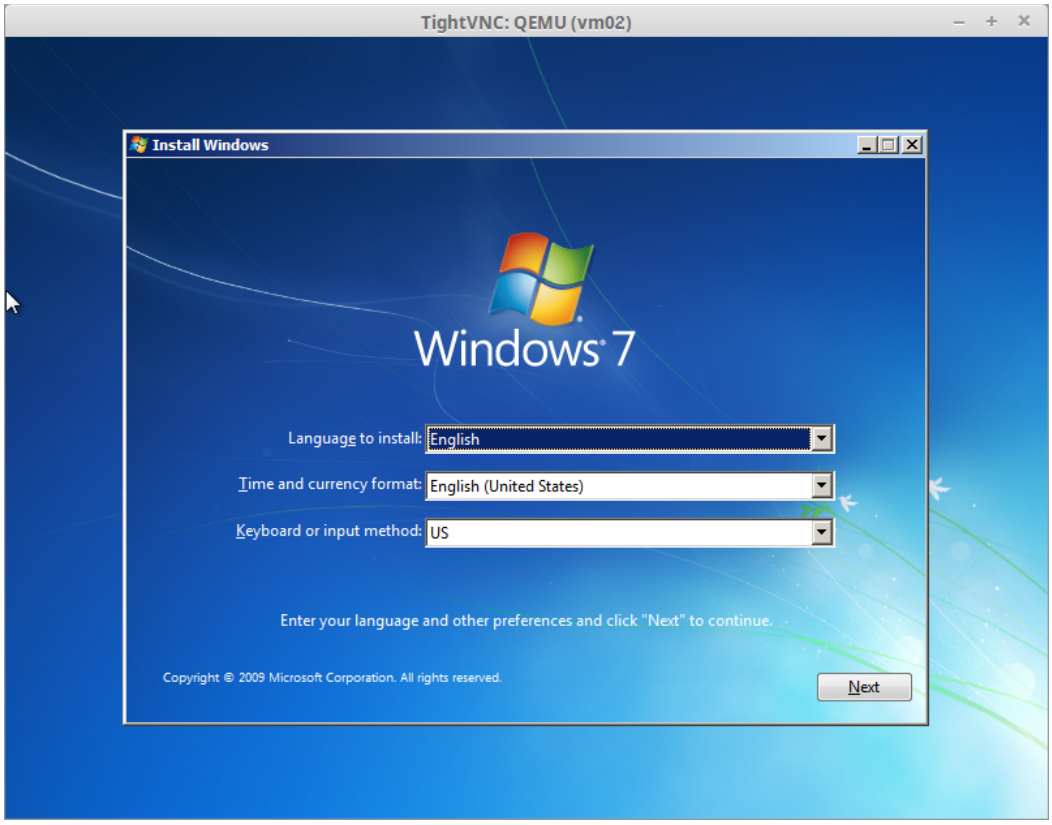

Illustration 7: Windows guest through VNC viewer

9. Manage and operate the domains

9.1 Check domain list

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

virsh # list
 Id    Name                           State
----------------------------------------------------
 1     vm01                           running
 2     vm02                           running
 3     vm03                           running

virsh # list --all
 Id    Name                           State
----------------------------------------------------
 1     vm01                           running
 -     vm02                           shut off
 3     vm03                           running
```

Some notes on possible states.

- **running** - domains which are currently active on a CPU.
- **blocked** / **blocking** - domain is presently idle, waiting for I/O or waiting for the hypervisor.
- **paused** - domain is suspended.
- **shutdown** - domains in the process of shutting down.
- **Domains** - domain is off and not using system resources.
- **crashed** - domain failed while running and is no longer running.

Displaying domain information

```
virsh # dominfo vm01
Id:             2
Name:           vm01
UUID:           b4a8695a-ccbb-49b1-9acd-cd4f649285ba
OS Type:        hvm
State:          running
CPU(s):         2
CPU time:       244.3s
Max memory:     524288 KiB
Used memory:    524288 KiB
Persistent:     yes
Autostart:      disable
Managed save:   no
Security model: none
Security DOI:   0
```

9.2 Shutdown a domain

```
virsh # shutdown vm01
Domain vm01 is being shutdown
```

9.3 Reboot a domain

```
virsh # reboot vm01
Domain vm01 is being rebooted
```

9.4 Terminate a domain

Destroying a domain is an immediate ungraceful shutdown and stops any guest domain sessions. Use the *destroy* option only when the guest is non-responsive.

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

virsh # destroy vm01
Domain vm01 destroyed
```

9.5 Start a domain

```
virsh # start vm01
Domain vm01 started
```

9.6 Suspend a domain

A running domain maybe suspended. The domain is kept in memory but it will not be scheduled any more and therefore it still consumes system RAM. Disk and network I/O will not occur while the guest is suspended. This operation is immediate and the guest must be restarted with the *resume* option.

```
virsh # suspend vm02
Domain vm02 suspended

virsh # list --all
 Id    Name                           State
----------------------------------------------------
 1     vm01                           running
 2     vm02                           paused
 3     vm03                           running
```

9.6.1 Resume a domain

```
virsh # resume vm02
Domain vm02 resumed

virsh # list --all
 Id    Name                           State
----------------------------------------------------
 1     vm01                           running
 2     vm02                           running
 3     vm03                           running
```

9.7 Save a domain

Saving a domain stops the guest VM and saves the data to a file, which may take some time given the amount of memory in use by the domain. The state of the guest can be recovered with the *restore* option.

```
virsh # save vm02 vm02file.vmsav

Domain vm02 saved to vm02file.vmsav

virsh # list --all
 Id    Name                           State
----------------------------------------------------
 1     vm01                           running
 3     vm03                           running

/virt/kvm/images$ ls
vm01.img  vm02file.vmsav  vm02.img
```

9.7.1 Restoring a domain

To restore a guest that was previously saved.

```
virsh # restore /virt/kvm/images/vm02file.vmsav
Domain restored from /virt/kvm/images/vm02file.vmsav

virsh # list --all
 Id    Name                           State
----------------------------------------------------
 1     vm01                           running
 2     vm02                           running
 3     vm03                           running
```

9.8 Cloning a domain

First the domain must be *suspended* or *shutdown* prior to cloning.

```
virsh # suspend vm01
Domain vm01 suspended

virsh # exit

$ virt-clone --connect qemu:///system \
             --original vm01 \
             --name cloned-debian-vm \
             --file /virt/kvm/images/cloned-debian-vm

WARNING  Setting the graphics device port to autoport, in order to
avoid conflicting.
Allocating 'cloned-debian-vm'        |  12 GB     00:49

Clone 'cloned-debian-vm' created successfully.
```

Now that it is created check it is there and run it.

```
virsh # list --all
 Id    Name                           State
----------------------------------------------------
 1     vm01                           paused
 2     vm02                           running
 3     vm03                           running
 -     cloned-debian-vm               shut off

virsh # start cloned-debian-vm
Domain cloned-debian-vm started

virsh # list --all
 Id    Name                           State
----------------------------------------------------
 1     vm01                           paused
 2     vm02                           running
 3     vm03                           running
 4     cloned-debian-vm               running
```

Now resume the original *vm01* and check that all four domains are in fact running.

```
virsh # resume vm01
Domain vm01 resumed

virsh # list --all
 Id    Name                           State
----------------------------------------------------
 1     vm01                           running
 2     vm02                           running
 3     vm03                           running
 4     cloned-debian-vm               running
```

9.9 Delete a domain

Firstly *destroy*, then *undefine* it and *vol-delete* the associated volume to completely remove the domain VM.

```
virsh # destroy vm01
Domain vm01 has been destroyed

virsh # undefine vm01
Domain vm01 has been undefined

virsh # vol-delete /virt/kvm/images/vm01.img
Vol /virt/kvm/images/vm01.img deleted
```

9.10 Domain XML file

This XML file defines the detail of the domain VM. It is possible to edit this file to implement change and reload the domain to incorporate these changes. As an example change the name from *vm01* to *vm01_Debian_8* and reload.

```
$ virsh --connect qemu:///system dumpxml vm01 > vm01.xml

$ cat vm01.xml
<domain type='kvm' id='1'>
  <name>vm01</name>
  <uuid>b4a8695a-ccbb-49b1-9acd-cd4f649285ba</uuid>
  <memory unit='KiB'>524288</memory>
  <currentMemory unit='KiB'>524288</currentMemory>
  <vcpu placement='static'>2</vcpu>
  <resource>
    <partition>/machine</partition>
  </resource>
  <os>
    <type arch='x86_64' machine='pc-i440fx-2.1'>hvm</type>
    <boot dev='hd'/>
  </os>
  <features>
    <acpi/>
    <apic/>
    <pae/>
  </features>
  <cpu mode='custom' match='exact'>
    <model fallback='allow'>Haswell</model>
  </cpu>
  <clock offset='utc'>
    <timer name='rtc' tickpolicy='catchup'/>
    <timer name='pit' tickpolicy='delay'/>
    <timer name='hpet' present='no'/>
  </clock>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/bin/kvm</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2'/>
      <source file='/virt/kvm/images/vm01.snapshot01'/>
      <backingStore type='file' index='1'>
        <format type='raw'/>
        <source file='/virt/kvm/images/vm01.img'/>
        <backingStore/>
      </backingStore>
      <target dev='vda' bus='virtio'/>
      <alias name='virtio-disk0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
    </disk>
```

```
      <disk type='block' device='cdrom'>
        <driver name='qemu' type='raw'/>
        <backingStore/>
        <target dev='hda' bus='ide'/>
        <readonly/>
        <alias name='ide0-0-0'/>
        <address type='drive' controller='0' bus='0' target='0' unit='0'/>
      </disk>
      <controller type='usb' index='0' model='ich9-ehci1'>
        <alias name='usb0'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x7'/>
      </controller>
      <controller type='usb' index='0' model='ich9-uhci1'>
        <alias name='usb0'/>
        <master startport='0'/>
                  <address  type='pci'  domain='0x0000'  bus='0x00'  slot='0x04'  function='0x0'
multifunction='on'/>
      </controller>
      <controller type='usb' index='0' model='ich9-uhci2'>
        <alias name='usb0'/>
        <master startport='2'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x1'/>
      </controller>
      <controller type='usb' index='0' model='ich9-uhci3'>
        <alias name='usb0'/>
        <master startport='4'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x2'/>
      </controller>
      <controller type='pci' index='0' model='pci-root'>
        <alias name='pci.0'/>
      </controller>
      <controller type='ide' index='0'>
        <alias name='ide0'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x1'/>
      </controller>
      <interface type='bridge'>
        <mac address='52:54:00:c5:50:05'/>
        <source bridge='br0'/>
        <target dev='vnet0'/>
        <model type='virtio'/>
        <alias name='net0'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
      </interface>
      <serial type='pty'>
        <source path='/dev/pts/0'/>
        <target port='0'/>
        <alias name='serial0'/>
      </serial>
      <console type='pty' tty='/dev/pts/0'>
        <source path='/dev/pts/0'/>
        <target type='serial' port='0'/>
        <alias name='serial0'/>
      </console>
      <input type='tablet' bus='usb'>
        <alias name='input0'/>
      </input>
      <input type='mouse' bus='ps2'/>
      <input type='keyboard' bus='ps2'/>
      <graphics type='vnc' port='5900' autoport='yes' listen='127.0.0.1'>
        <listen type='address' address='127.0.0.1'/>
      </graphics>
      <video>
        <model type='cirrus' vram='9216' heads='1'/>
        <alias name='video0'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
      </video>
      <memballoon model='virtio'>
        <alias name='balloon0'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
      </memballoon>
    </devices>
  </domain>
```

Using the sed stream editor, make the change and backup the original file.

```
$ sed -i.bak 's/<name>vm01</<name>vm01_Debian_8</' vm01.xml
```

Confirm the change.

```
$ diff vm01.xml vm01.xml.bak
2c2
<   <name>vm01_Debian_8</name>
---
>   <name>vm01</name>
```

undefine the old domain to prevent an error resulting from a duplicate UUID.

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

virsh # undefine vm01
Domain vm01 has been undefined

virsh # destroy vm01
virsh # start vm01_Debian_8

virsh # define vm01.xml
Domain vm01_Debian_8 defined from vm01.xml

virsh # start vm01_Debian_8
Domain vm01_Debian_8 started
```

9.11 Creating a snapshot

A snapshot is a copy of the domain disk file at a given point in time. Snapshots are used to restore a domain to a particular point in time when a failure or system error occurs.

There are two classes of snapshots for QEMU guests. Internal snapshots are contained completely within a qcow2 file but have the drawback of slow creation times, less maintenance by upstream QEMU, and the requirement to use QCOW2 disks.

External snapshots are more useful, because they work with any type of original disk image, can be taken without guest downtime. They are created when using the *virsh snapshot-create-as --disk-only* option. Unfortunately until improvements are made in *libvirt*, snapshots can only be made, they cannot be restored through *libvirt*. However there is a workaround using the XML file for the domain.

Here are the domains running

```
virsh # list
 Id    Name                 State
------------------------------------
 1     vm01                 running
 2     vm02                 running
 3     vm03                 running
```

Looking at the first domain *vm01*, confirm that there are no existing snapshots.

```
virsh # snapshot-list vm01
 Name                 Creation Time             State
------------------------------------------------------------
```

Using *virsh* create a snapshot, note that the snapshot will be in QEMU Copy On Write (QCOW) format and not RAW.

```
virsh # snapshot-create-as --domain vm01 --name "snap01-vm01"
        --disk-only --atomic
        --diskspec vda,file=/virt/kvm/images/snap01-vm01.img
Domain snapshot snap01-vm01 created
```

- **--diskspec** - adds the disk elements to the Snapshot XML file.
- **--disk-only** - takes the snapshot of only the disk.
- **--atomic** - either the snapshot is run completely or fails w/o making any changes.

Review the new snapshot.

```
virsh # snapshot-list vm01
 Name                 Creation Time             State
------------------------------------------------------------
 snap01-vm01   2016-01-11 12:14:10 +0000 disk-snapshot
```

Review the files in the directory and the file types. Note that the snapshot is of QCOWv3 format.

```
$ ls /virt/kvm/images/*vm01*
/virt/kvm/images/snap01-vm01.img  /virt/kvm/images/vm01.img

$ sudo file /virt/kvm/images/vm01.img
/virt/kvm/images/vm01.img: DOS/MBR boot sector

$ sudo file /virt/kvm/images/snap01-vm01.img
/virt/kvm/images/snap01-vm01.img: QEMU QCOW Image (v3), has backing
file (path /virt/kvm/images/vm01.img), 12582969344 bytes

$ sudo qemu-img info /virt/kvm/images/vm01.img
image: /virt/kvm/images/vm01.img
file format: raw
virtual size: 12G (12582969344 bytes)
disk size: 12G
```

```
$ sudo qemu-img info /virt/kvm/images/snap01-vm01.img
image: /virt/kvm/images/snap01-vm01.img
file format: qcow2
virtual size: 12G (12582969344 bytes)
disk size: 5.7M
cluster_size: 65536
backing file: /virt/kvm/images/vm01.img
backing file format: raw
Format specific information:
    compat: 1.1
    lazy refcounts: false
```

Confirm the image that the domain is running currently.

```
virsh # domblklist vm01
Target     Source
------------------------------------------------
vda        /virt/kvm/images/snap01-vm01.img
hdc        -
```

Make some changes to the rumming domain. Add a file for example.

```
vm01~$ echo 'I am a file added to the snapshot' >> ~/snapshot.txt
```

9.12 Revert to a previous snapshot

Unfortunately the virsh command *snapshot-revert vm01 snap01-vm01.img* is not yet working with KVM, however it is quite simple to make the revert.

Shutdown the domain, a simple shutdown of the guest OS is fine. And confirm it is infact not running.

```
virsh # list --all
 Id    Name                 State
----------------------------------
 2     vm02                 running
 3     vm03                 running
 -     vm01                 shut off
```

Edit the domain XML file to change the name of the image back to the original.

```
virsh # edit vm01
```

Change the source file to the original image and change the type to raw to match the original image.

```
....
....

  <disk type='file' device='disk'>
    <driver name='qemu' type='qcow2'/>
    <source file='/virt/kvm/images/snap01-vm01.img'/>
    <target dev='vda' bus='virtio'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
  </disk>
....
....
```

to:

```
....
....
  <disk type='file' device='disk'>
    <driver name='qemu' type='raw'/>
    <source file='/virt/kvm/images/vm01.img'/>
    <target dev='vda' bus='virtio'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
  </disk>
....
....
```

The change is confirmed back to the console.

```
Domain vm01 XML configuration edited.
```

Now start the domain.

```
virsh # start vm01
Domain vm01 started
```

Check the running domain block device. It has reverted to the original image.

```
virsh # domblklist vm01
Target     Source
------------------------------------
vda        /virt/kvm/images/vm01.img
hdc        -
```

Check and confirm that the *snapshot.txt* file is there.

```
vm01~$ ls snapshot.txt
ls: cannot access snapshot.txt: No such file or directory
```

9.13 Delete a snapshot

If a snapshot is to be deleted, remove the metadata defining it and then delete the file.

```
virsh # snapshot-delete --domain vm01 --metadata "snap01-vm01"
Domain snapshot snap01-vm01 deleted

$ sudo rm /virt/kvm/image/snap01-vm01.img
```

10. Virtual networks

It is possible to build virtual networks within the KVM Hypervisor and even link them to the outside physical network. By way of explanation I will go through the steps to build the following network. All elements are in fact virtual. *vm01* will act as a router and will have access to the physical world via a bridged connection to *br0* which has a connection to *eth0* on the host computer. Like has been shown already this receives an IP address from a DHCP Server on the network. The router will have a second virtual interface *eth1* which is connected to a virtual network *prtbr0* and has a static IP address 10.1.1.1/24 which will act as an IP gateway for the other guests.

A Windows 7 guest *vm02*, and two Debian 8 guests *vm03* plus *vm04* are configured with their primary interfaces connected to the *prtbr0* virtual network, with IP addresses 10.1.1.2, 10.1.1.3 and 10.1.1.4 respectfully. Each has 10.1.1.1 as its gateway.

The router *vm01* is configured to forward IP traffic and has a *iptables* masquerade rule to give the guests access to the Internet in the physical world via *Network Address Translation (NAT)*.

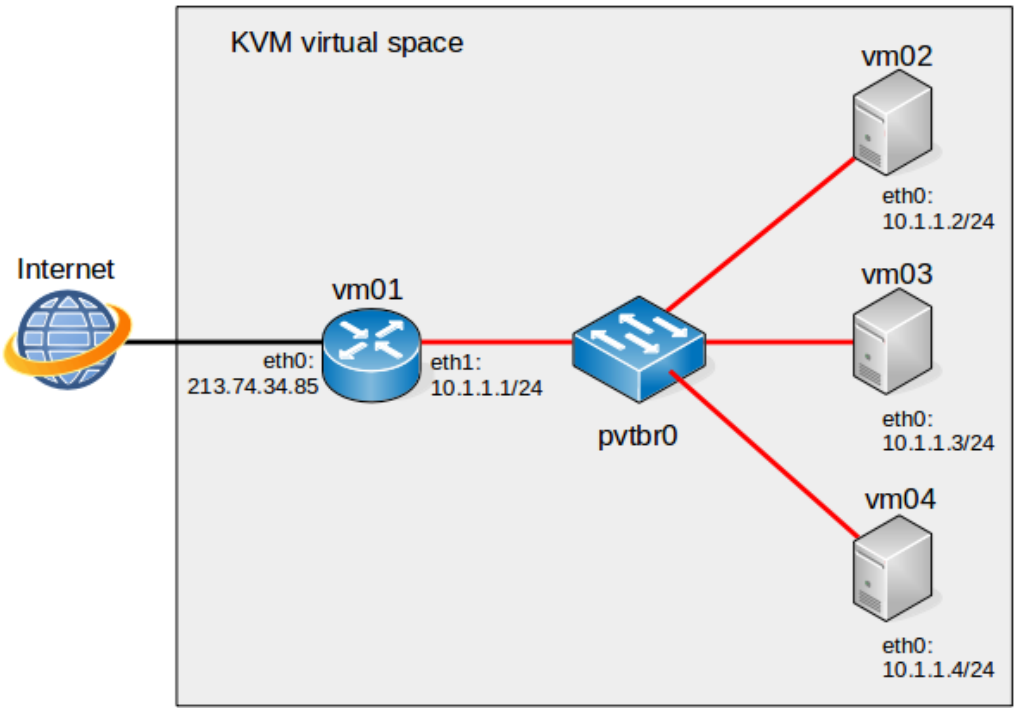

Illustration 8: Virtual network

10.1 Create a new network

With a number of guest domains on a KVM host it may become desirable to have a virtual network. List existing networks, presuming there is only one at this stage.

```
$ virsh --connect qemu:///system net-list --all
 Name                 State      Autostart     Persistent
----------------------------------------------------------
 default              inactive   no            yes
```

Create a new network XML file.

```
$ sudo bash -c 'cat << EOM >> /virt/kvm/networks/pvtnet01.xml
<network>
   <name>pvtnet01</name>
   <bridge name="pvtbr0" />
</network>
EOM'

$ cat /virt/kvm/networks/pvtnet01.xml
<network>
   <name>pvtnet01</name>
   <bridge name="pvtbr0" />
</network>
```

Define the network in the KVM hypervisor.

```
virsh # net-define /virt/kvm/networks/pvtnet01.xml
Network pvtnet01 defined from /virt/kvm/networks/pvtnet01.xml

virsh # net-list --all
 Name                 State      Autostart     Persistent
----------------------------------------------------------
 default              inactive   no            yes
 pvtnet01             inactive   no            yes
```

start and *enable* the automatic startup of the new virtual network.

```
virsh # net-start pvtnet01
Network pvtnet01 started

virsh # net-autostart pvtnet01
Network pvtnet01 marked as autostarted

virsh # net-list --all
 Name                 State      Autostart     Persistent
----------------------------------------------------------
 default              inactive   no            yes
 pvtnet01             active     yes           yes
```

The bridge created can be seen on the on the hypervisor host OS.

```
$ sudo brctl show
bridge name     bridge id               STP enabled     interfaces
br0             8000.305a3a083921       no              eth0
                                                        vnet2
pvtbr0          8000.5254002db430       yes             pvtbr0-nic
                                                        vnet0
```

10.2 Configure domain VMs to connect to the new network

Edit each domain *vm02*, *vm03* and *vm04* by replacing the existing *<interface>* as shown. A choice of *nano* or *VIM* editors is given the first time the *edit* command is used. Here is an example of one domain being configured.

```
virsh # shutdown vm03
Domain vm03 is being shutdown

virsh # domstate vm03
shut off

virsh # edit vm03
```

Change:

```
....
....
  <interface type='bridge'>
    <mac address='52:54:00:c5:50:05'/>
    <source bridge='br0'/>
    <model type='virtio'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
  </interface>
....
....
```

to this:

```
....
....
  <interface type='network'>
    <source network='pvtnet01'/>
    <model type='virtio'/>
  </interface>
....
....
```

and upon saving the change will be confirmed. Looking at the file again it can be seen that the *MAC address* and *address type* are automatically reconfigured.

```
Domain vm03 XML configuration edited
```

Note: For Microsoft Windows use *<model type='rtl8139'/>* instead of *<model type='virtio'/>*

Restart *vm03*.

```
virsh # start vm03
Domain vm03 started
```

10.2.1 IP configuration for guests

10.2.1.1 Debian guest

Configure an IP address on Debian guest *vm03* and *vm04*, edit each to have a similar look to the example below which reflects the file for *vm03*.

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

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
        address 10.1.1.3
        netmask 255.255.255.0
        network 10.1.1.0
        broadcast 10.1.1.255
        gateway 10.1.1.1
        dns-nameservers 8.8.8.8

:wq!
```

Restart the *networking* service.

```
$ sudo /etc/init.d/networking restart
[ ok ] Restarting networking (via systemctl): networking.service.
```

Confirm the IP settings.

```
$ ip addr list
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
default qlen 1000
    link/ether 52:54:00:6c:df:6c brd ff:ff:ff:ff:ff:ff
    inet 10.1.1.3/24 brd 10.1.1.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::5054:ff:fe6c:df6c/64 scope link
       valid_lft forever preferred_lft forever
```

Test between each guest once completed. First from *vm03* to *vm04*.

```
$ ping -c1 10.1.1.4
PING 10.1.1.3 (10.1.1.4) 56(84) bytes of data.
64 bytes from 10.1.1.4: icmp_seq=1 ttl=64 time=0.128 ms

--- 10.1.1.4 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.139/0.139/0.139/0.000 ms
```

and now from *vm04* to *vm03*.

```
$ ping -c1 10.1.1.3
PING 10.1.1.3 (10.1.1.3) 56(84) bytes of data.
64 bytes from 10.1.1.3: icmp_seq=1 ttl=64 time=0.139 ms

--- 10.1.1.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.139/0.139/0.139/0.000 ms
```

10.2.1.2 Windows guest

Configure an IP address on the *vm02* guest.

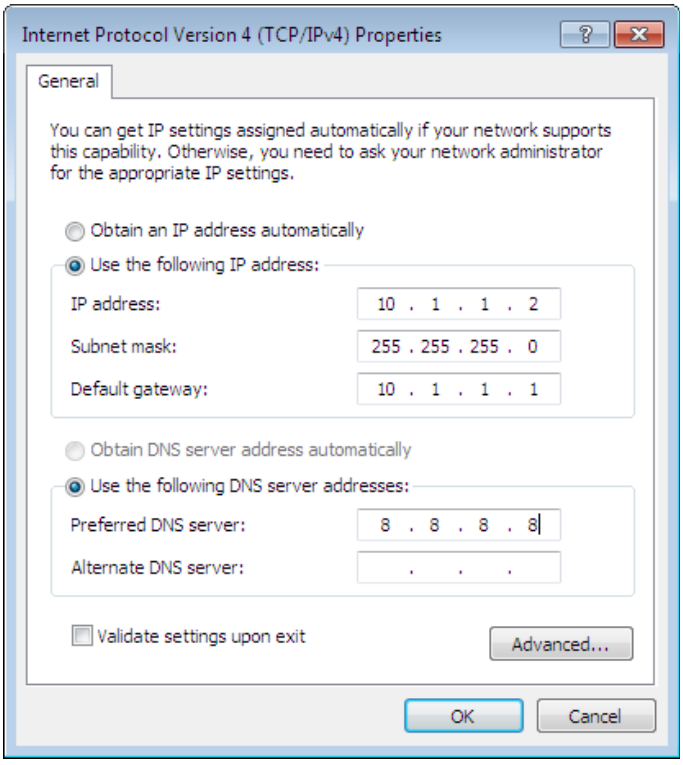

Illustration 9: Windows IPv4 address assignment

For testing purposes allow *ICMPv4 ECHO_REQ* on the Windows guest.

Control Panel --> **System and security** --> **Windows Firewall**
Advanced settings --> **Inbound rules** --> **New rule** --> **Custom rule**

Protocol and ports: Protocol: **ICMPv4**

--> **Customize**,
--> Choose **Specific ICMP types**, check the box **echo request**.

Open a command prompt and ping the domains *vm03* and *vm04* just configured.

```
C:> ping -n 1 10.1.1.3

Pinging 10.1.1.3 with 32 bytes of data:
Reply from 10.1.1.3: bytes=32 time<1ms TTL=64

Ping statistics for 10.1.1.3:
     Pakets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
     Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:> ping -n 1 10.1.1.4

Pinging 10.1.1.4 with 32 bytes of data:
Reply from 10.1.1.4: bytes=32 time<1ms TTL=64

Ping statistics for 10.1.1.4:
     Pakets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
     Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

10.3 Configure the router vm01-Debian_8

10.3.1 Establish forwarding

By default the Debian 8 guest acting as a router *vm01* has IP forwarding disabled.

```
$ cat /proc/sys/net/ipv4/ip_forward
0
```

Enable forwarding by switching the value to 1.

```
$ sudo bash -c 'echo 1 > /proc/sys/net/ipv4/ip_forward'
```

This interface configuration is similar to the guests except that two interfaces are required; one for the *Internet* and the other to the *pvtnet01*.

```
virsh # shutdown vm01
Domain vm01 is being shutdown

virsh # domstate vm01
shut off
```

```
virsh # edit vm01
```

Change:

```
....
....
  <interface type='bridge'>
    <mac address='52:54:00:c5:50:05'/>
    <source bridge='br0'/>
    <model type='virtio'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
  </interface>
....
....
```

to this:

```
....
....
  <interface type='bridge'>
    <mac address='52:54:00:c5:50:05'/>
    <source bridge='br0'/>
    <model type='virtio'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
  </interface>
  <interface type='network'>
    <source network='pvtnet01'/>
    <model type='virtio'/>
  </interface>
....
....
```

```
Domain vm01 XML configuration edited.

virsh # start vm01
```

10.3.2 Configure the interfaces on the router guest

Edit the */etc/network/interfaces/* file as follows:

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

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary interface eth0
auto eth0
iface eth0 inet dhcp

# The secondary interface eth1 connected to pr
auto eth1
iface eth1 static
        address 10.1.1.1
        netmask 255.255.255.0
        network 10.1.1.0
        broadcast 10.1.1.255
        dns-nameservers 8.8.8.8

:wq!
```

Restart the *networking* service.

```
$ sudo /etc/init.d/networking restart
[ ok ] Restarting networking (via systemctl): networking.service.
```

Confirm the IP settings.

```
~$ ip addr list
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
default qlen 1000
    link/ether 52:54:00:fe:26:b7 brd ff:ff:ff:ff:ff:ff
    inet 213.74.34.85/24 brd 213.74.34.255 scope global dynamic eth0
       valid_lft 429sec preferred_lft 429sec
    inet6 fe80::5054:ff:fefe:26b7/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
default qlen 1000
    link/ether 52:54:00:3c:a8:87 brd ff:ff:ff:ff:ff:ff
    inet 10.1.1.1/24 brd 10.1.1.255 scope global eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::5054:ff:fe3c:a887/64 scope link
       valid_lft forever preferred_lft forever
```

Inspect the routes table.

```
$ ip route list
default via 213.74.34.1 dev eth0
default via 213.74.34.1 dev eth0  proto static  metric 1024
10.1.1.0/24 dev eth1  proto kernel  scope link  src 10.1.1.1
213.74.34.0/24 dev eth0  proto kernel  scope link  src 213.74.34.85
```

Ping hosts on the *pvtnet0*.

```
$ ping -c1 10.1.1.2
PING 10.1.1.2 (10.1.1.2) 56(84) bytes of data.
64 bytes from 10.1.1.2: icmp_seq=1 ttl=128 time=0.502 ms

--- 10.1.1.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.502/0.502/0.502/0.000 ms

$ ping -c1 10.1.1.3
PING 10.1.1.3 (10.1.1.3) 56(84) bytes of data.
64 bytes from 10.1.1.3: icmp_seq=1 ttl=64 time=0.264 ms

--- 10.1.1.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.264/0.264/0.264/0.000 ms

$ ping -c1 10.1.1.4
PING 10.1.1.4 (10.1.1.4) 56(84) bytes of data.
64 bytes from 10.1.1.4: icmp_seq=1 ttl=64 time=0.169 ms

--- 10.1.1.4 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.169/0.169/0.169/0.000 ms
```

Ping a public IP address.

```
$ ping -c1 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=56 time=28 ms

--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 26.562/26.562/26.562/0.000 ms
```

10.3.3 Configure IP Masquerade on the router

The IP Masquerade is a resource used so that a set of machines may use a single IP address. This permits the hidden nodes on a private network, such as 10.1.1.0/24 to access the public network; but they cannot directly accept external connections; only through the machine that has the real IP. Traffic from the private network to the Internet must have the private source IP address replaced with the Masquerade public IP address. Outward connections must be tracked so incoming returning traffic can be correctly identified and the correct private IP address swapped in the packet header for the public IP address before forwarding to the private network. This is achievable because of a GNU/Linux feature called Connection Tracking (conntrack). While on the public network the source IP address is masqueraded as if it came from the GNU/Linux server.

```
$ sudo iptables --table nat \
                --append POSTROUTING \
                --source 10.1.1.0/24 \
                --out-interface eth0 \
                --jump MASQUERADE
```

10.4 Review the bridge control on the host

Have a look at what has happened on the host.

```
$ sudo brctl show
bridge name   bridge id              STP enabled     interfaces
br0           8000.305a3a083921      no              eth0
                                                     vnet0
                                                     vnet1
                                                     vnet5
pvtbr0        8000.5254002db430      yes             pvtbr0-nic
                                                     vnet2
                                                     vnet3
                                                     vnet4
                                                     vnet6
```

10.5 Confirm that the guests have Internet access

Confirm each of the domain guests have Internet access through *vm01*.

10.5.1 vm02

```
C:> ping -n 1 8.8.8.8

Pinging 8.8.8.8 with 32 bytes of data:
Reply from 8.8.8.8: bytes=32 time<1ms TTL=64

Ping statistics for 8.8.8.8:
     Pakets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
     Minimum = 25ms, Maximum = 25ms, Average = 25ms
```

10.5.2 vm03

```
$ ping -c1 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=55 time=18.8 ms

--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 18.844/18.844/18.844/0.000 ms
```

10.5.3 vm04

```
$ ping -c1 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=55 time=25.2 ms

--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 25.204/25.204/25.204/0.000 ms
```

11. Graphical management

Now that the basics are mastered it is time to check out the KVM graphical manager.

```
$ virt-manager
```

This can be run on the hypervisor or remotely where it will connect to the hypervisor using *ssh*. When ran initially on a machine that is not the hypervisor it shows *localhost (QEMU) - Not Connected.* This simply demonstrates that there is no hypervisor on the local machine. To connect to the hypervisor select *File --> Add Connection...*.

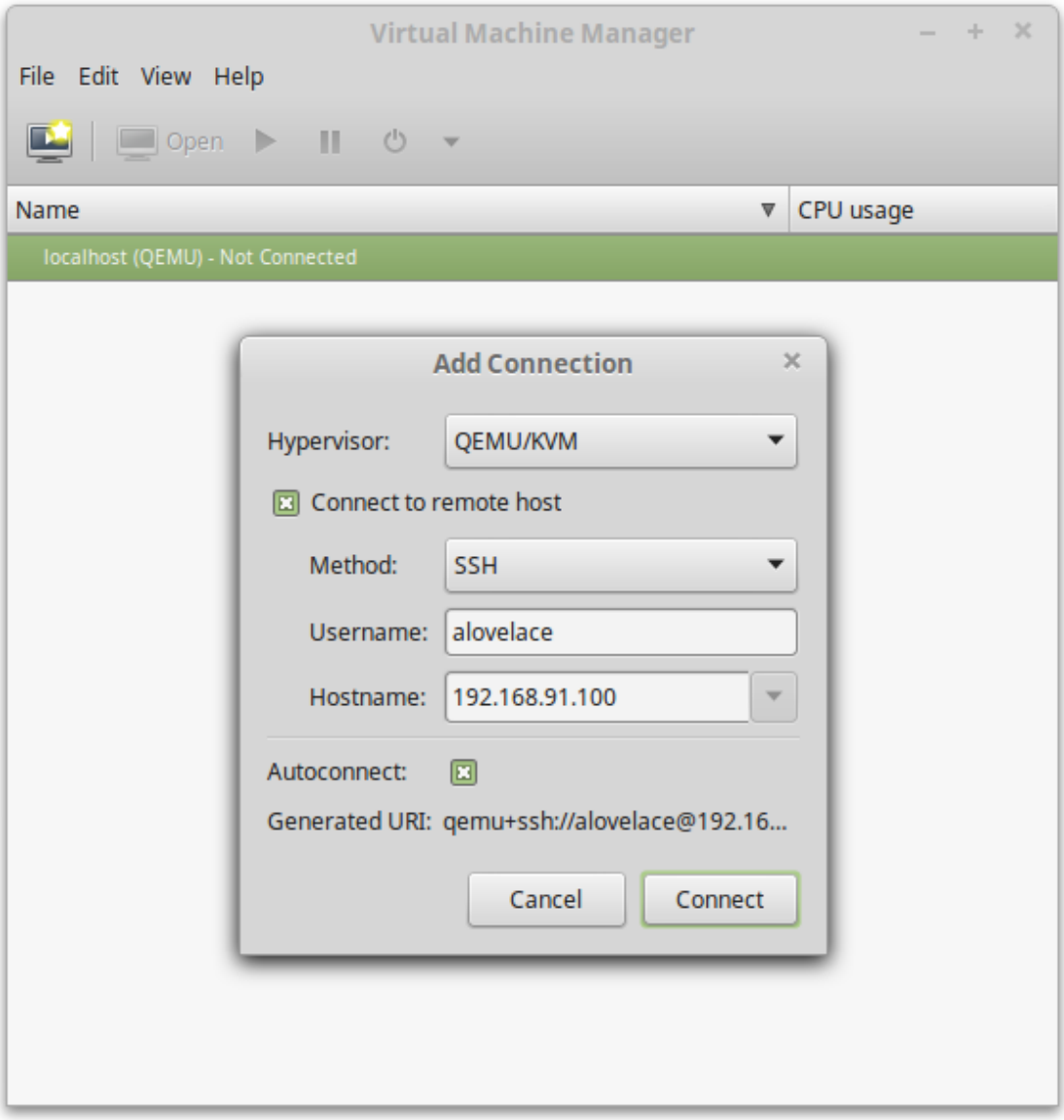

Illustration 10: Hypervisor 'Add connection' screen

Once logged in all of the domains are displayed in the initial window with the CPU usage graph for each.

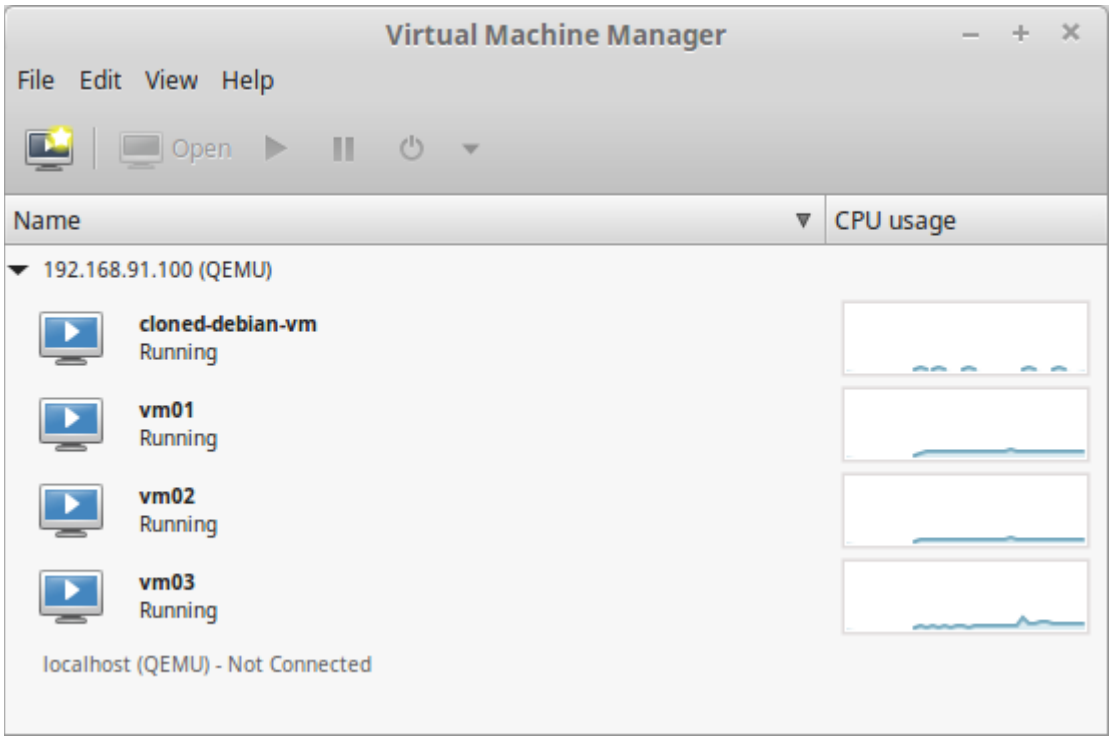

Illustration 11: Domain dashboard view

For each it is possible with a right-mouse click for each individual domain to:

- Pause.
- Shutdown.
- Migrate.
- Open.

Select *Open* or double click on the domain to run a viewer to the domain VM in a similar fashion to that with *virt-viewer*. If the *View --> Details* is selected it is possible to adjust the domain options which will become active after the next domain reboot.

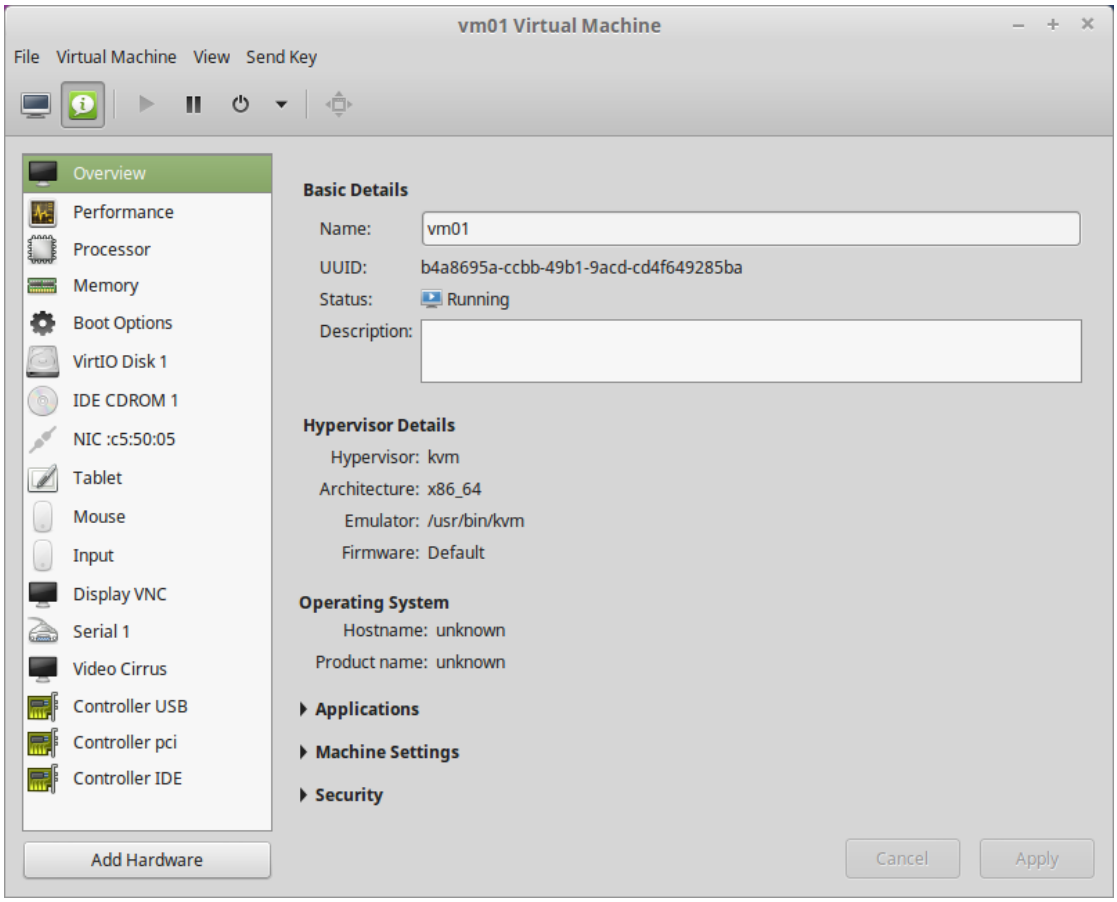

Illustration 12: Domain details

12. Importing image from other virtualisation platform

What if an image is received from another platform. For example an Open Virtualisation Archive (.ova) file from Oracle VirtualBox. Well the OVA file is actually a tar archive and can be extracted to show the Virtual Machine Disk (.vmdk) file within.

For KVM it is better to use the QCOW2 format.

Extract the files from the .ova file.

```
$ tar -xvf vm.ova
vm.ovf
vm.vmdk

$ qemu-img convert -O qcow2 vm-disk1.vmdk vm-disk1.qcow2
vm-disk1.qcow2   vm-disk1.vmdk   vm.ova   vm.ovf

$ cp vm-disk1.qcow2 /virt/kvm/images
```

If the details of the original VM are not readily available like CPU, RAM etc. they can be found in the Open Virtualisation Format (OVF) description file vm-vbox.ovf.

12.1 Install the KVM Domain with the import file

The existing disk that was created on another virtualisation platform can be imported when building the KVM domain. Build the KVM domain with the existing disk indicated in the disk path. The comma delimited options indicate the device is of the type disk and the disk device driver is *virtio*.

```
$ virt-install \
--connect qemu:///system \
--virt-type=kvm \
--name vm-OVA \
--ram 2048 \
--vcpus=2 \
--disk path=/virt/kvm/images/vm-disk1.qcow2,device=disk,bus=virtio \
--graphics vnc,listen=0.0.0.0 \
--noautoconsole \
--os-type linux \
--os-variant debianwheezy \
--network=bridge:br0 \
--import

Starting install...
Creating domain...                    |    0 B     00:01
Connected to domain  vm-OVA

Escape character is ^]

Domain creation completed. You can restart your domain by running:
  virsh --connect qemu:///system start vm-OVA
```