**Virtualization Stack- Layers**
VMs
libvirtd (to manage hypervisor and VMs (KVM or otherwise), LXC containers, etc) - generic interface for virtualization that can be used by various management programs like virt-manager GUI, virsh virtualization shell, virt-install is just an installation-base which can be done in virt-manager)
QEMU (manage/intercept VM's virtual instructions through hypervisor to physical CPU)
KVM (adds virtual to physical base) - kvm, kvm-intel, kvm-amd modules at kernel level - kernel support
Hardware [run egrep 'vmx | svm' /proc/cpuinfo <---to check virtualization and KVM support]

Hypervisor: virt SW "deployed directly on bare-metal host machines" like KVM
- enables mapping between physical CPUs and virtual CPUs for HW acceleration
- Once KVM is set up, QEMU (quick emulator) manages the execution of the mapping
Libvirt is the standard SW to manage the VMs and hypervisor

**Starting up, check things and do basics:**
In order to run virtualization infrastructure like KVM, the CPU has to support HW virtualization
lscpu to check. VT-x (or vmx) for Intel, AMD-V (or svm) for AMD. "grep vmx /proc/cpuinfo" also works.
**egrep 'vmx | svm' /proc/cpuinfo** <---check cpu virtualization support
**lsmod | grep kvm** <---check if you have the kernel modules available
**systemctl status libvertd** <---check if you have the Libvirt services running
kvm, kvm-intel, kvm-amd modules
**ip link show** would likely show vibr0, the virtual bridge, your embedded switch
*libvertd used by virt-manager, virt-install and virsh. Also central to other KVM work*
Some other things to check
You need a 64-bit kernel - run **"arch"** to find something like **"x86_64"**

Enter the virsh shell by just typing **"virsh"**
Typing **help** gives you the lon categorized list of command

Outside of the shell, just precede commands with "virsh"
virsh list (shows ID, name, and state of what you have)
virsh list --all
virsh destroy NAME -- shuts off the machine, won't show in list, but list --all will, and it says NAME is "shut off"
virsh start NAME -- will start it up again - note that it says "domain" started

Configuration files live in /etc/libvirt/
libvirt.conf libvirtd.conf lxc.conf qemu.conf qemu-lockd.conf virtlockd.conf
Directories nwfilter qemu and storage

Here we focus on the Qemu directory - it is an old emulator but adopted in the KVM environment
in our sample, it just has networks (a directory) and small.xml (for the VM)
Opening the xml in vim we see a warning at the top that the file is autogenerated and should properly edited using the **virsh edit** command "or other program using the libvirt API"
Doing so basically ensures that nothing else will (likely) be writing to the same file while it is being edited

This file uses an image file for the disk which is noted is not for best possible performance, but is the easy way. The alternative to an image file is not mentioned, but I assume it means a real device.

When the VM is running, **ip link show** displays both vibr0 and a vnet01 interface, the latter being the virtual NIC for the VM.
Default both are in NAT mode. Details not specified

**Virt-manager** GUI is mostly like what you get with VirtualBox. L-ctrl L-Alt to exit mouse pointer from VM window

yum install -y kvm libvirt virt-manager qemu-kvm
if you need to make an ISO of your install disk, it is something like dd if=/dev/sr0 of=/root/rhel7.iso
All the given exercise was, was using virt-manager GUI for everything. virsh destroy NAME to shut down the machine. If virsh destroy doesn't work and you still see it with virsh list --all as running, it may be because virsh can't stop a machine that is starting up it's kernel inits

---

RHEL7 installer has 4 virtualization subgroups:
Virt. hypervisor group: includes libvert and qemu-kvm packages
Virt. Client: support to install/ manage VMs. virsh, virt-install, virt-manager, virt-top and virt-viewer
Virt. Tools: for offline management of VMs, including libguestfs
Virt. Platform: Interface to access and control VMs. libvert, libvert-client and virt-who packages
If not already installed, just run "yum -y group install "virtualization hypervisor" "virtualization client" etc...
Vugt says you can do this: "yum install -y kvm libvirt virt-manager qemu-kvm"

**Virtual Network Switch and Interface**
Represented as virbr0 in ip addr show with an IP address of 192.168.122.1/24, NAT masq'd
libvirt uses dnsmasq for virtual switch DNS and DHCP
libvert supports two additional switch configurations: routed and isolated. As implied, isolated can't communicate outside host environment, and routed can.
Individual VM's will show up with names like vnet0

Storage pool and storage volume contains space for VMs to collectively used. Can be a file, file system in a logical volume, partition or entire disk; can also be remote via NFS or iSCSI. Once defined, can be sliced up into storage volumes and assigned to VMs as block storage devices.
Example shown had pool type of "filesystem directory" set at /var/lib/libvert/images/default

**Libvert's GUI Virtual Machine Manager (virt-manager)**
Virt-install and virsh are more extensive command-line equivalents
- provisioning and installation of guest OS's (attended/unattended, local/remote)
- management and monitoring of both hypervisor and VMs
- wizards for configuration of VMs, virtual networks, storage pools
- option to start functionality at host boot time or not
- multiple virtual networks are supported

More options - vmx option in CPU settings to embed VMs in VMs (Vugt)

**virsh Command Structure**
virsh commands follow the format:
# virsh subcommand domain-id | name | uuid [options]
*See Virsh Commands Brief document for overview of subcommands.*

**Virsh subcommand categories: Quick view**

General subcommands: cd, pwd, connect, hostname, sysinfo, nodeinfo, list
Device: detach/attach-interface, -disk
Network: net-create, net-autostart ... all begin with net-
Network Interface: iface--bridge, all begin with iface-
Storage Pool: pool-build, all begin with pool-
Storage Volume: vol-name, vol-path
*Common among these: -create/-delete, -define/-undefine, -destroy, -dumpxml, -edit, -info, -list, -start*
Domain commands have some of these common ones (minus the prefix) and include desc, domdisplay, domhostname, domhost, dominfo, domstate, reboot/shutdown, suspend/resume, vcpuinfo

virsh --- opens a virsh terminal - too many commands to mention here. List with 'help'
virsh list <----running VMs
virsh list --all <---all VMs
virsh destroy small <--- immediately shuts off virsh
virsh start small <----start it up again
/etc/libvert/ contains the conf info, also contains the qemu directory
In the qemu configs- small.xml - Best way to edit these files is with the virsh shell
(virsh edit small instead of vim small.xml)
Holds basic VM configuration items (RAM, CPU and other stuff)

Excursion into comparative virtualization:
"Para-virtualized means the operating system is modified to be aware that it is running inside a virtual machine, and makes hypercalls directly (ABI), rather than issuing privileged instructions.  Xen boots from a bootloader such as GNU GRUB, and then usually loads a paravirtualized host operating system into the host domain (dom0). Xen supports five different approaches to running the guest operating system: HVM (hardware virtual machine), HVM with PV drivers, PVHVM (HVM with PVHVM drivers), PVH (PV in an HVM container) and PV (paravirtualization).  A PV system is simpler to manage and reduces the attack surface exposed to potentially malicious guests.

***Hypervisor and "Virtual Machine Manager" seem to be interchangable terms when talking about the general virtualization market***
2013 marketshare: VMware vSphere 56%, MS Hyper-V (2012 Server) 28%, Citrix XenServer 3.3%, KVM 4%, Oracle VirtualBox 0.5%

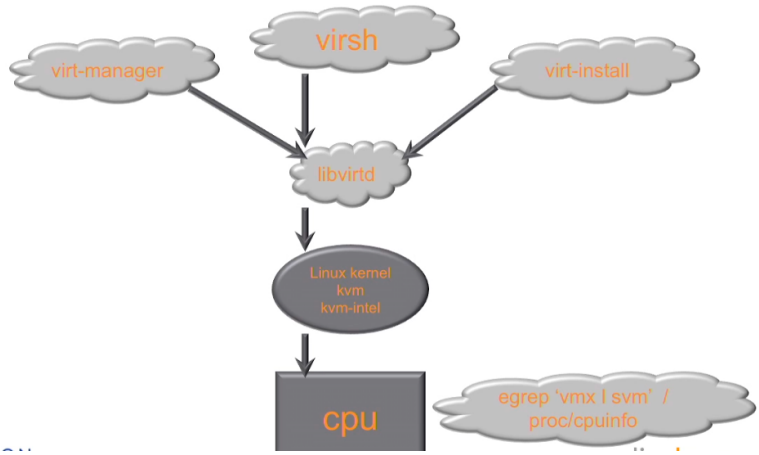
Oracle and KVM have about equal deployment,

"What is the difference between KVM and QEMU?
QEMU uses emulation; KVM uses processor extensions (HVM) for virtualization."

Parallels' Virtuozzo/OpenVZ NOT full virtualization.
While virtualization technologies like VMware and Xen provide full virtualization and can run multiple operating systems and different kernel versions, OpenVZ uses a single patched Linux kernel and therefore can run only Linux. All OpenVZ containers share the same architecture and kernel version. This can be a disadvantage in situations where guests require different kernel versions than that of the host.
OpenVZ allows a physical server to run multiple isolated operating system instances known as containers."
"However because it doesn't have the overhead of a true hypervisor it is very fast and efficient over Xen, KVM and VMware"

"XEN and KVM runs kernel insade VPS while OpenVZ runs containers using a shared kernel. This means that KVM VPS does not actually have the same amount of memory available to userspace that OpenVZ does. OpenVZ is an extremely lightweight virtualization technology that easily outperforms any full virtualization.
In short user with KVM/XEN 512 MB ram VPS doesn't have available same amount of free RAM like user with OVZ 512 MB RAM ,so basicly if you install just Centos 6 on 256MB Ram OVZ VPS and in same time install just centos 6 on 256 MB Ram KVM VPS you will have more free RAM on OVZ VPS no metter both have same amount of RAM and same OS ,like I say on KVM/XEN you run OS and kernel like on dedicated server while on OVZ kernel is shared and in practice not run in your VPS"

Linux-KVM.org




cat /proc/cpuinfo   look for vmx
lsmod | grp kvm
systemctl status libvirtd
ip link show -- look for br0
KVM needs 64bit cpu (x86_64)