Proprietary Statement
Copyright © 2012 Novell, Inc. All rights reserved.

Novell, Inc., has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed on the Novell Legal Patents Web page (http://www.novell.com/company/legal/patents/) and one or more additional patents or pending patent applications in the U.S. and in other countries.

No part of this publication may be reproduced, photocopied, stored on a retrieval system, or transmitted without the express written consent of the publisher.

Novell, Inc.
404 Wyman Street, Suite 500
Waltham, MA 02451
U.S.A.
www.novell.com
Novell Trademarks
For Novell trademarks, see the Novell Trademark and Service Mark list (http://www.novell.com/company/legal/trademarks/tmList.html).

Third-Party Materials
All third-party trademarks are the property of their respective owners.

Software Piracy
Throughout the world, unauthorized duplication of software is subject to both criminal and civil penalties.

If you know of illegal copying of software, contact your local Software Antipiracy Hotline. For the Hotline number for your area, access Novell’s World Wide Web page (http://www.novell.com) and look for the piracy page under “Programs.” Or, contact Novell’s anti-piracy headquarters in the U.S. at 800-PIRATES (747-2837) or 801-861-7101.

Disclaimer
Novell, Inc., makes no representations or warranties with respect to the contents or use of this documentation, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose.

Further, Novell, Inc., reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes. Further, Novell, Inc., makes no representations or warranties with respect to any software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc., reserves the right to make changes to any and all parts of Novell software, at any time, without any obligation to notify any person or entity of such changes.

Any products or technical information provided under this Agreement may be subject to U.S. export controls and the trade laws of other countries. You agree to comply with all export control regulations and to obtain any required licenses or classification to export, re-export or import deliverables. You agree not to export or re-export to entities on the current U.S. export exclusion lists or to any embargoed or terrorist countries as specified in the U.S. export laws. You agree to not use deliverables for prohibited nuclear, missile, or chemical biological weaponry end uses. See the Novell International Trade Services Web page (http://www.novell.com/info/exports/) for more information on exporting Novell software. Novell assumes no responsibility for your failure to obtain any necessary export approvals.

This Novell Training Manual is published solely to instruct students in the use of Novell networking software. Although third-party application software packages are used in Novell training courses, this is for demonstration purposes only and shall not constitute an endorsement of any of these software applications.

Further, Novell, Inc. does not represent itself as having any particular expertise in these application software packages and any use by students of the same shall be done at the student’s own risk.
Contents

SECTION 1: Introduction to KVM Virtualization 100
SECTION 2: Introduction to Virtual Machines 146
SECTION 3: Introduction to Open Source Virtualization Management 164
SECTION 4: Manage Virtualization Platforms with Libvirt 219
SECTION 5: Delegate Virtualization Administration 255
SECTION 6: Other Virtualization Management Tasks 287
SECTION 7: Configure Virtual Hardware for KVM Virtualization 287
Objectives

• KVM Terminology
• KVM Virtualization Architecture
• Virtualization Modes
• CPU Virtualization
• Introduction to KVM Hardware Virtualization
• KVM Feature and OS Support
KVM Terminology
Community Terminology

- **Domain**: A container for a running virtual machine. Colloquially, the virtual machine (VM) itself
- **Physical Driver**: A device driver that talks directly to the hardware
- **Virtual Driver**: A device driver in a VM that fulfills requests by going to the physical driver in the vHost
KVM Virtualization Architecture
With traditional virtualization, the hypervisor runs on top of a “Host” OS. This is also known as a Type II Hypervisor.
In a Type II Hypervisor the virtualization layer is responsible for mediation of access to the underlying hardware, sharing access to the hardware with virtual drivers, and VM management.
Xen Virtualization Architecture

With Xen, the hypervisor runs directly on top of the hardware and does not require a “Host OS”. This is also known as a Type I Hypervisor.

Xen is a “lean” hypervisor in that it is only responsible for mediation of access to the underlying hardware and not for sharing access to that hardware with virtual drivers. No device drivers are loaded into the Xen hypervisor, which makes it compatible with virtually any hardware platform.

Sharing of hardware devices with virtual drivers and VM management are handled by one of the virtual machines.
VMware ESX Virtualization Architecture

With VMware ESX, the hypervisor and Host OS are merged to create a “fat” hypervisor. The hypervisor/OS runs directly on top of hardware making it Type I as well. A “fat” hypervisor is both responsible for mediation of access to the underlying hardware but also of sharing that hardware with virtual drivers through hardware emulation and/or paravirtual APIs. This requires device drivers to be loaded into the hypervisor, which in turn limits its compatibility with hardware platforms.
Hyper-V Virtualization Architecture

With Hyper-V, the hypervisor is also considered Type I.
With KVM (or the Kernel Virtual Machine), a kernel module is loaded into the Linux kernel that turns it into a hypervisor. KVM would essentially be a Type I Hypervisor because it is running directly on top of the hardware.

With KVM, the Linux kernel becomes a “fat” hypervisor because it not only mediates access to the underlying hardware but also loads physical drivers and shares access to the underlying hardware devices with virtual drivers.

Device emulation and VM management are handled by a modified version of QEmu running in user space.
With container based virtualization, no hypervisor is involved. The “Host” OS (in this case Linux) provides all OS services to each virtual container.

Only a single kernel (Linux) can run on the hardware at a time with container based virtualization.
Virtualization Modes
Virtualization Modes

Full-virtual

- VMs can run native (unmodified) OS
- Requires the need to trap and emulate all privileged instructions*, or emulate everything
- Performance is negatively impacted
- KVM Requires a VT enabled CPU
- KVM is natively a full-virtual hypervisor

*Some processors are difficult to fully virtualize such as traditional x86

KVM is natively a “full-virtual” hypervisor but does not support native paravirtualization like Xen. KVM is really only the kernel space component with the virtualized hardware component being handled by a modified version of Qemu and virtio. Qemu can only provide hardware emulation where virtio can provide paravirtual hardware.
Virtualization Modes

Progressive Paravirtual

- Hybrid of Full and Paravirtualization
- OS is “Enlightened” to know about paravirtual hypercalls
- Some parts of the OS can use paravirtual hypercalls other parts must use trap and emulation
- Performance is improved
- Requires VT enabled CPU
- Linux can run progressively paravirtual in KVM if the kernel was compiled with Paravirtual Operations (PV-ops)

In a progressively paravirtual machine, both hardware emulation and paravirtual APIs are provided. Depending on the operating system running in this type of virtual machine, one or both types of hardware interaction may be used. Operating systems generally start booting using the hardware emulation but switch over to using the paravirtual APIs when they are detected. The paravirtual APIs that are exposed into a progressively paravirtual machine can differ depending on the virtualization platform. KVM exposes the generic paravirtual API of the Linux kernel named Paravirt-ops (PV-ops). With the VMware hypervisor, these paravirtual APIs are named VMI and also leverage the Paravirt-ops in the Linux kernel. In the case of the Xen hypervisor, an adapter has been created that translates the native Xen paravirtual APIs into Hyper-V paravirtual APIs which are then exposed into the virtual machine. The native Xen APIs are not exposed into the VM however. The native Xen APIs are not needed because anything ported to the Xen hypervisor, can run in a native paravirtual machine. The Hyper-V hypervisor has a similar adapter that translates its native paravirtual APIs into the Xen APIs and exposes both the Xen and its native paravirtual APIs into the VM.
CPU Virtualization
CPU Virtualization

Privileged Rings – Traditional Hypervisor

- OS Kernel and parts of the Hypervisor run in Ring 0
- Ring 0 and 1 of the VM are emulated in Ring 1

In traditional hypervisors, the kernel and parts of the hypervisor are loaded into Ring0, or the ring of greatest privilege. Ring 0 and Ring 1 of the virtual machine are often emulated in the “real” Ring 1 by the hypervisor, while Rings 2 & 3 of the VM run in the “real” Rings 2 & 3. Emulation of rings 0 & 1 requires extra code and complexity in the hypervisor and possibly overhead in performance.
CPU Virtualization: VT vs Non-VT

Pseudo Privileged Rings – Full-virtual OS

- VT CPUs create pseudo privileged rings into which KVM creates a virtual machine

KVM virtual machines require the VT extensions available in modern CPUs (both Intel and AMD). When creating a virtual machine, the KVM hypervisor, being prompted by qemu-kvm, requests that the VT enabled CPU create a pseudo set of privileged ring. KVM creates the virtual machine in these rings. The kernel of an OS running in this type of VM is loaded into memory and executed in this new “pseudo” Ring0. All interactions with hardware are then mediated by the KVM and the Linux kernel running in the “real” Ring0. Leveraging the VT instructions in the CPU allows the hypervisor to be less complex and offload some of the virtualization workload to the hardware.
KVM vHost Boot Process
Native OS Boot Process

Native OS Boot Process – Step 1

- Boot Loader loads OS Kernel into in Ring 0
- kvm and platform specific kvm_ kernel modules load

Just like traditional OSes, the first phase of the boot process is to load the kernel into Ring0 and have it initialize all of the hardware. KVM is not a separate binary from the Linux kernel as is the case with the Xen hypervisor. The kvm.ko kernel module loads causing the platform specific (Intel vs AMD) kvm kernel module to load (kvm_intel.ko or kvm_amd.ko). The Linux kernel can now act as a hypervisor.
Native OS Boot Process

Native OS Boot Process – Step 2

- OS Kernel run in Ring 0
- The libvirt daemon starts in user space (Ring 3)

The second phase of the boot process is to load the virtualization management services (libvirt) along with any other applications into Ring3. The interface for managing the Linux/KVM hypervisor is now in place and virtual machines can be launched.
KVM vHost Boot Process
Full-virtual Machine Boot Process

VM Boot Process: Step 1

- The VM's configuration file is read by libvirt
- Libvirt formulates a qemu-kvm command that begins the VM creation process

Notes:
Full-virtual Machine Boot Process

VM Boot Process: Step 2

- The hypervisor first requests a new set of pseudo rings from the VT enabled CPU

Notes:
Full-virtual Machine Boot Process

VM Boot Process: Step 3

- A virtual machine is created in the new pseudo set of rings based on the parameters in the VM's configuration
- At this point the VM is in a paused state

Notes:
Full-virtual Machine Boot Process

VM Boot Process: Step 4

- The qemu-dm and virtio set up the virtualized hardware layer
- The qemu-kvm unpauses the VM into real mode
- A virtual BIOS is loaded into the VM by the qemu-dm and runs, discovering all virtual/physical hardware
Full-virtual Machine Boot Process

VM Boot Process: Step 5

- The virtual BIOS loads the boot loader from the MBR/boot sector of the disk into the VM's memory
Full-virtual Machine Boot Process

VM Boot Process: Step 6

- The boot loader loads the kernel of the OS into the VM's memory and the kernel initializes the virtual/physical hardware and switches to protected mode
- The OS continues to boot as normal in the VM
Introduction to KVM Hardware Virtualization
The KVM hypervisor will run on any x86 or x86_64 hardware that has the VT extensions enabled.
The KVM hypervisor will run on any x86 or x86_64 hardware that has the VT extensions enabled.
The Qemu-dm manages all hardware emulation. The virtio or vhost subsystem provides an optimized (paravirtual) I/O path for the virtual machines.
The Qemu-dm also handle all physical device passthrough. PCI device passthrough also requires IOMMU to be enabled in the chipset.

KVM supports PCI virtual function passthrough in the same manner as physical device passthrough if the device (i.e. NIC) have SR-IOV support. SR-IOV also requires IOMMU support in the chipset.
KVM OS and Feature Support
## Hardware Support / Requirements

### Linux/KVM Hypervisor Memory and CPU Support

<table>
<thead>
<tr>
<th>VM Server Limits</th>
<th>x86*</th>
<th>x86 PAE*</th>
<th>x86_64</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUs</td>
<td>1 – 32</td>
<td>1 – 32</td>
<td>1 – 4096</td>
</tr>
<tr>
<td>Physical Memory</td>
<td>512MB – 4GB</td>
<td>512-MB-64GB</td>
<td>512MB-16TB</td>
</tr>
</tbody>
</table>

### VM Memory and CPU Support (Supported by SUSE)

<table>
<thead>
<tr>
<th>VM Server Limits</th>
<th>x86*</th>
<th>x86 PAE*</th>
<th>x86_64</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCPUs per VM</td>
<td>1 – 4</td>
<td>1 – 8</td>
<td>1 – 8</td>
</tr>
<tr>
<td>Memory per VM</td>
<td>128MB – 2GB</td>
<td>128-MB-16GB</td>
<td>128MB-32GB</td>
</tr>
</tbody>
</table>

*Only the x86_64 version of KVM will be supported by SUSE*
OSes Supported by SUSE Running in a KVM Virtual Machine

Supported* on KVM:
- SLES10 (latest SP)
- SLES11 (latest SP)
- SLES 9 (latest SP)
- RHEL 4 & 5 (& 6?)
- Windows Server 2003 & 2003R2
- Windows Server 2008 & 2008R2
- Windows 2000 Server
- Windows XP
- Windows Vista
- Windows 7

* See /usr/share/doc/packages/kvm/kvm-supported.txt for the lastest information about supported OSes

Notes:
LAB 1-1: Install a Virtualization Host Server

**Summary:** In this exercise, you install the packages requires to have a Xen and a KVM virtualization host server.

**Special Instructions**
Use the following values in the exercise:

*(none)*

**Duration:** ? min.

Lab Notes:
Fundamentals of Virtualization with SUSE Linux Enterprise
Section 2: Introduction to Virtual Machines
Objectives

- The Anatomy of a Virtual Machine
- Xen Virtual Machine Configuration Information
- KVM Virtual Machine Configuration Information
- Introduction to Virtual Disks
The Anatomy of a Virtual Machine
The Anatomy of a VM

What makes up a Virtual Machine?

• **Configuration**
  - For the VM server to launch a VM, it must know the parameters to use when launching the VM

• **Disk(s)**
  - The VM's disk(s) store the OS and data used by the VM
  - The VM's disk(s) can be simple disk images or physical disks

• **Snapshots / Checkpoint files / etc.**

Notes:
KVM Virtual Machine Configuration Information
KVM VM Configuration Information

2 Ways to Store VM Configuration

- **Unmanaged VMs**
  - VMs require a VM configuration file to be launched
  - The configuration file can exist anywhere on disk
    - (recommended to be in same directory as disk image)
- **Managed VMs**
  - VMs have a copy of their configuration file stored in the libvirt “database” `/etc/libvirt/qemu/`
    - VM configuration information exists **only** in the Libvirt database of the vHost on which they have been installed or are running

Notes:
KVM VM Configuration Information

Example: XML VM Configuration

```xml
<domain type="kvm" id="-1">
  <name>vm1</name>
  <memory>52428</memory>
  <maxmem>1024</maxmem>
  <vcpus>1</vcpus>
  <uuid>3e2ba59a-1ce9-4046-1e13-3b1ec2d363f2</uuid>
  <on_crash>destroy</on_crash>
  <on_reboot>restart</on_reboot>
  <on_poweroff>destroy</on_poweroff>
  <bootloader>/usr/lib/xen/boot/domUloader.py</bootloader>
  <bootargs>--entry=xvda2:/boot/vmlinuz-xen,/boot/initrd-xen</bootargs>
  <os>
    <type arch='x86_64' machine='pc'>hvm</type>
    <boot dev='hd' />
  </os>
  <devices>
    <input type="mouse" bus='xen'/>
    <graphics type="vnc" port='-1'/>
    <interface type="ethernet">
      <target dev='vif-1.1'/>
      <mac addres='00:16:3e:3d:8f:a6'/>
    </interface>
    <disk type='file' device='disk'>
      <driver name='file'/>
      <source file="/var/lib/xen/images/vm1/disk0.img'/>
      <target dev='xvda'/>
    </disk>
  </devices>
</domain>
```

Notes:
Introduction to Virtual Disks
Virtual Disks

Virtual Disk Types

- **Physical Disks**
  - Any block device in /dev
    - Disks
    - Partitions
    - Logical volumes

- **Disk Images**
  - raw disk images
  - ISO images
  - Qemu qcow2
  - VMware vmdk
  - Microsoft vhd

- **SAN Disks**
  - iSCSI
  - NPIV Fibre Channel

Notes:
Objectives

● Virtualization Management Layers and Virtualization Platform Stacks
● Virtual Machine Management Tools
● Virtual Machine Creation with vm-install
Virtualization Management Layers and Virtualization Platform Stacks
Identity:
This is the layer where identity, authentication for and secure communication with the virtualization management utilities is administered.

Data Center Automation:
This is the layer that higher level utilities that help automate the virtualization stack and virtual machines running on that stack function. Tasks such as providing high availability, automatic migration based on system utilization, and virtual machine lifecycle management happen here.

Virtualization Management:
This is the layer that utilities that manage the virtualization stack as a whole function. Tasks such as starting and stopping virtual machines from stored configuration, viewing the status of virtual machines, connection to and managing virtual machines and delegation of such tasks happen here.

Hypervisor Management:
This is the layer where low level utilities that interact with and manage the hypervisor and virtualized hardware function. The low level task such as domain (virtual machine) creation and destruction, domain suspend, resume and migration and virtual device creation, attaching and detaching happen here.

Virtualized Hardware:
This is the layer where the subsystems that virtualize or emulate the hardware for a virtual machine function. Tasks such as block and network I/O virtualization and PCI bus and device emulation happen here.
**Hypervisor:**
This layer is where the entity that does the actual "virtualization" functions. Tasks such as CPU virtualization and memory management happen here.
Virtual Machine Management Tools
VM Management Tools

Built-in Virtualization Management Tools

- `xm / xl` (Xen)
- `vm-install` (Xen & KVM)
- `libvirt, virt-manager, virsh, virt-viewer, libvirt-cim` (Xen & KVM)

Notes:
VM Management Tools

Accessing Xen Management Tools

- **YaST->Virtualization**

![YaST Control Center]

Notes:
Virtual Machine Creation with vm-install
VM Management Tools

Built-in Virtualization Management Tools

- `xm / xl` (Xen)
- `vm-install` (Xen & KVM)
- `libvirt, virt-manager, virsh, virt-viewer, libvirt-cim` (Xen & KVM)

All tools mentioned here are available in SLES11. They are simply organized here by the main developer/maintainer of the tool.
The vm-install Utility

vm-install

- Simplified tool to create and launch installations into Xen VMs
- Can be used to create/install both Paravirtual and Full-virtual VMs
- Creates both a VM configuration file and registers the VM as a managed VM in the xenstore database
- Can be used to perform Physical to Virtual migrations by importing existing disks or disk image files
- Can be run with or without a GUI
- Accessible as a stand alone application, from within virt-manager or through YaST

YaST->Virtualization->Create Virtual Machines

Notes:
The vm-install Utility

vm-install

• Introduction Screen

Notes:
The vm-install Utility

vm-install

- Choose the OS

Notes:
The `vm-install` Utility

`vm-install`

- VM Installation
- Configuration Summary
- Fully-virtualized VM

Notes:
LAB 6-2: Install a SLES11 KVM Virtual Machine

Summary: In this exercise you will install SLES 11 as a virtual machine in KVM.

Special Instructions:

\[ INSTALL\_URL=(provided\ by\ instructor) \]

Duration: 30 min.

Lab Notes:
Fundamentals of Virtualization with SUSE Linux Enterprise
Section 4: Manage Virtualization Platforms with Libvirt
Objectives

- Introduction to Libvirt
- Libvirt Virtualization Management Utilities
- Use Libvirt Virtualization Management Utilities Remotely
Introduction to Libvirt
VM Management Tools

Built-in Virtualization Management Tools

- `xm / xl` (Xen)
- `vm-install` (Xen & KVM)
- `libvirt, virt-manager, virsh, virt-viewer, libvirt-cim` (Xen & KVM)

All tools mentioned here are available in SLES11. They are simply organized here by the main developer/maintainer of the tool.
Introduction to Libvirt

What is libvirt?

- libvirt is a virtualization abstraction layer that sits between the hypervisor and the management tools
- Provides a uniform and extensible virtualization management api
- Provides management interface to multiple hypervisors
  - Xen, KVM, Qemu, VirtualBox, LCX, etc.

Notes:
Introduction to Libvirt

libvirt management tools

- Has multiple front ends for interaction
  - Daemon (libvirtd)
  - GUI (virt-manager, virt-viewer)
  - CLI (virsh)
  - CIM (libvirt-cim)

Notes:
## Important Files and Directories (1/4)

### Main Configuration Files and Directories

<table>
<thead>
<tr>
<th>File/Dir</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc/libvirt/</td>
<td>-Main libvirt configuration directory</td>
</tr>
<tr>
<td>/etc/libvirt/libvirtd.conf</td>
<td>-Main configuration file for the libvirt daemon</td>
</tr>
<tr>
<td>/etc/libvirt/qemu.conf</td>
<td>-Configuration file for the qemu driver</td>
</tr>
<tr>
<td>/etc/libvirt/lxc.conf</td>
<td>-Configuration file for the LXC driver</td>
</tr>
<tr>
<td>/etc/libvirt/qemu/</td>
<td>-Managed configuration directory for KVM VMs</td>
</tr>
<tr>
<td>/etc/libvirt/storage/</td>
<td>-Managed configuration directory for storage</td>
</tr>
<tr>
<td>/var/lib/libvirt/</td>
<td>-Directory for transient libvirt configuration</td>
</tr>
</tbody>
</table>

Notes:
## Important Files and Directories (2/4)

### /etc/libvirt/

<table>
<thead>
<tr>
<th>File/Dir</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qemu/autostart/</td>
<td>-Directory for VM's config file that are to be started automatically by libvirt</td>
</tr>
<tr>
<td>qemu/networks/</td>
<td>-Directory for virtual network definition files</td>
</tr>
<tr>
<td>qemu/networks/autostart/</td>
<td>-Directory for virtual network definition files that should be started automatically</td>
</tr>
<tr>
<td>storage/</td>
<td>-Directory for storage pool definition files</td>
</tr>
<tr>
<td>storage/autostart/</td>
<td>-Directory for storage pool definition files that should be started automatically</td>
</tr>
</tbody>
</table>

Notes:
### Important Files and Directories (3/4)

**/var/lib/libvirt/**

<table>
<thead>
<tr>
<th>File/Dir</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boot/</td>
<td>-Directory for transient</td>
</tr>
</tbody>
</table>
| dnsmasq/  | -Directory for virtual network DHCP leases files  
            -named `NET_NAME.leases` |
| images/   | -Generic directory for disk image files  
            -A Libvirt storage pool is defined and active for this directory by default |
| libxl/    | -Directory for transient Libxenlight files |
| lxc/      | -Directory for transient LXC driver files |
| network/  | -Directory for transient virtual network files  
            -a copy of the network's XML config file goes here while the network is started |
| qemu/     | -Directory for transient QEMU driver files  
            -QEMU VM monitor socketed files  
            -dump, save an snapshot images |

**Notes:**
The libvirt Daemon

libvirtd

- Provides an interface to libvirt both locally and remotely
- Supports remote connections via TLS, SSH or TCP via SASL
  - Local URI: xen:///l, qemu:///l, etc.
  - TLS remote URI: xen:///REMOTE_HOST/
  - qemu:///REMOTE_HOST/system
  - SSH remote URI: xen+ssh://user@REMOTE_HOST/
  - qemu+ssh://user@REMOTE_HOST/system
  - TCP remote URI: xen+tcp://user@REMOTE_HOST/
  - qemu+tcp://user@REMOTE_HOST/system

- Configuration file:
  - /etc/libvirt/libvirtd.conf

Notes:
Libvirt Virtualization Management Utilities
The virsh Command

What is the virsh command?

- Provides a CLI interface to libvirt to manage virtual machines
- Similar capabilities to the xm command
- Can connect to a remote host via a TLS or SSH tunnel

Notes:
The virsh Command

The virsh command syntax:

- **Querying, Connecting & Debugging**
  - **Querying**
    - `virsh list` List running Domains (VMs)
    - `virsh dumpxml VM_NAME` List configuration for a domain in XML
  - **Other Information about VMs**
    - `virsh dominfo VM_NAME or VM_ID` Print information about the domain
  - **Interactive Shell**
    - `virsh domstate` Returns the state of a running domain
    - `virsh` Launch an interactive shell that accepts virsh commands without prefacing them with “virsh”
    - `quit` exits the shell

Notes:
The virsh Command

The virsh command syntax:

- **Querying, Connecting & Debugging**
  
  **Querying**
  
  `virsh list`  
  List running Domains (VMs)
  
  `virsh dumpxml VM_NAME`  
  List configuration for a domain in XML
  
  **Other Information about VMs**
  
  `virsh dominfo VM_NAME or VM_ID`  
  Print information about the domain
  
  `virsh domstate`  
  Returns the state of a running domain

Notes:
The virsh Command

The virsh command syntax:

- **Managing VM configuration**
  - Importing into 'xenstore'
    
    `virsh define XML_CONFIG_FILE`
    
    Imports a VM's config file into 'xenstore' making it a managed VM
  
  - Exporting out of 'xenstore'
    
    `virsh dumpxml VM_NAME > file`
    
    Exports VM configuration from 'xenstore' into a file (while VM is powered off)
  
  - Removing from 'xenstore'
    
    `virsh undefine VM_NAME`
    
    Removes a VM's information from 'xenstore'

Notes:
The virsh Command

The virsh command syntax:

- **Starting VMs**
  - Starting Unmanaged VMs
    - `virsh create XML_CONFIG_FILE`
      - Launch an unmanaged VM using configuration located in a stand alone XML configuration file
  - Starting Managed VMs
    - `virsh start VM_NAME`
      - Launch a managed VM using the VM's configuration located in 'xenstore'

Notes:
The virsh Command

The virsh command syntax:

- **Stopping VMs**
  - Stopping
    - `virsh shutdown VM_NAME` or `VM_ID`
      - Sends the 'shutdown -h now' command to the console of a VM bringing it down softly
    - `virsh reboot VM_NAME` or `VM_ID`
      - Sends the 'shutdown -r now' command to the console of a VM rebooting it softly
    - `virsh destroy VM_NAME` or `VM_ID`
      - Immediately stops and removes a VM
        - like pulling the power plug

Notes:
The virsh Command

The virsh command syntax:

- **VM Server Information**
  
  Information about the VM Server
  
  `virsh nodeinfo` List information about the VM Server node
  
  `virsh capabilities` List information about the hypervisor and node capabilities in XML format

Notes:
The virsh Command

The virsh command syntax:

- **Getting help**
  
  *Manual Page*
  
  `man virsh`  The virsh manual page
  
  *Quick Help*
  
  `virsh help`  Detailed help on the xm command

Notes:
LAB 8-1: Use Common virsh Commands

Summary: In this exercise, you use common virsh commands to manage virtual machines.

Special Instructions:

VM1_NAME=sles11sp2-kvm
VM2_NAME=w2k8r2-kvm

Duration: 10 min.

Lab Notes:
The virt-viewer Utility

What is the virt-viewer utility?

- Virt-viewer is vnc viewer for connecting to the GUI console of VMs that is independent of virt-manager
  - Built on the GTK vncviewer
  - Can lock pointer into VM window
  - Can send pre defined keystrokes into the VM

- Virt-viewer can connect to VMs on the local VM Server or on remote VM Servers
The virt-viewer Utility

**virt-viewer viewing running VMs**

---

**Notes:**
LAB 8-2: Use virt-viewer to Connect to a Running VM

Summary: In this exercise, you use the virt-viewer utility to connect to the GUI console of a running virtual machine.

Special Instructions:

PV_VM_NAME=sles11sp2-xen
PV_VM_CONFIG=/vmstore/sles11sp2-xen/sles11sp2-xen.xml

Duration: 10 min.
The virt-manager Utility

What is virt-manager?

- Provides a graphical interface to libvirt to manage Xen virtual machines
  - Start / Stop / Pause a Virtual Machine
  - Provide an overview of the Virtual Machine
  - Provide access to Hardware information and the ability to configure VM hardware
    - # of VCPUs
    - Memory allocation
    - block devices
  - Launch vm-install to configure / install new Xen Virtual Machines

Notes:
The virt-manager Utility

Virt-Manager - Host and VM View

Notes:
The virt-manager Utility

Virt-Manager – VM Details

Notes:
The virt-manager Utility

virt-manager with running VM

Notes:
LAB 8-3: Use virt-manager to Manage Virtual Machines

**Summary:** In this exercise, you use the virt-manager utility to manage local virtual machines.

**Special Instructions:**

**Duration:** 10 min.
Use Libvirt Virtualization Management Utilities Remotely
Access Libvirt Remotely

Supports remote connections via TLS, SSH or TCP via SASL

- SSH remote URI:  
  `xen+ssh://user@REMOTE_HOST/`  
  `qemu+ssh://user@REMOTE_HOST/system`

- TLS remote URI:  
  `xen://REMOTE_HOST/`  
  `qemu://REMOTE_HOST/system`

- TCP remote URI:  
  `xen+tcp://user@REMOTE_HOST/`  
  `qemu+tcp://user@REMOTE_HOST/system`

Notes:
LAB 8-4: Configure Key based Authentication in OpenSSH

**Summary:** In this exercise you generate a SSH keypair and then upload the public key to your lab partner's machine to allow you to log in remotely without a password or passphrase.

**Special Instructions:**

`REMOTE_IP`=(provided by instructor)

**Duration:** 10 min.

Lab Notes:
LAB 8-5: Use virsh to Connect to a Remote VM

Summary: In this exercise you use the virsh utility to connect to a remote VM.

Special Instructions:

REMOTE_IP=(provided by instructor)
VM_NAME=(provided by instructor)
VM_CONFIG_FILE=(provided by instructor)

Duration: 10 min.
LAB 8-6: Use virt-viewer to Connect to a Remote VM

Summary: In this exercise, you use the virt-viewer utility to connect to a remote VM.

Special Instructions:
- $REMOTE_IP$=(provided by instructor)
- $VM_NAME$=(provided by instructor)
- $VM_CONFIG_FILE$=(provided by instructor)

Duration: 10 min.

Lab Notes:
LAB 8-7: Manage VMs on a Remote vHost with virt-manager

Summary: In this exercise, you use the virt-manager utility to connect to the libvirt daemon on a remote vHost and manage its virtual machines.

Special Instructions:

VHOST_IP = (provided by instructor)

Duration: 10 min.

Lab Notes:
Fundamentals of Virtualization with SUSE Linux Enterprise
Section 5: Delegate Virtualization Administration
Objectives

- Introduction to the Delegation of Virtualization Administration
- Delegate Virtualization Administration with PolicyKit
- Delegate Virtualization Administration with UNIX Group Membership
- Delegate Virtualization Administration with SASL
Introduction to the Delegation of Virtualization Administration
Delegate Virtualization Administration

Xen with xm or xl

- Delegation can only be done via sudo

Xen or KVM with Libvirt

- Delegation can be done via:
  - PolicyKit
  - UNIX group membership
  - SASL

Notes:
Delegate Virtualization Administration with Libvirt
Introduction to PolicyKit

What is PolicyKit?

- **PolicyKit is:**
  - an application-level toolkit for defining how unprivileged processes to speak to privileged processes
  - a framework for centralizing the decision making process for granting access to privileged operation for unprivileged applications

- **PolicyKit specifically targets applications in rich desktop environments but also applies at the command line**
- **PolicyKit does not rely on kernel special features**

Notes:
### Introduction to PolicyKit

#### How Can Privilege Be Granted Traditionally?

<table>
<thead>
<tr>
<th>Method</th>
<th>Details</th>
</tr>
</thead>
</table>
| **SUID/SGID** | - No authentication required  
  - Entire application runs with privilege |
| **su** | - Root password required to authenticate  
  - Entire shell and all its child processes run with privilege |
| **sudo** | - User’s password required to authenticate  
  - Entire application runs with privilege |

#### Methods of Granting Privilege

**SUID/SGID**

With SUID/SGID, a bit is flipped in the POSIX permissions that say that no matter who runs the application, the application runs with root privileges. The entire application runs with root privileges. Delegation of privilege is done by adding users to a group who has permission to execute the application.

**su**

With su, the user wishing to execute an application with root privilege must provide the root password before (or while) executing the application. The entire application runs with root privileges. Delegation of privilege requires the root password to be entrusted to any user who must have elevated privilege.

**sudo**

With sudo, the application is executed by the user using the sudo utility. The sudo utility check for granted privilege and then executes the application with elevated privilege on behalf of the user. The entire application then runs with root privileges. Delegation of privilege is done by adding the applications that need to be executed with privilege to the sudoers file along with users who need to execute these applications. These users must provide their own password to authenticate when executing these application with sudo.
### Introduction to PolicyKit

#### How Can Privilege Be Granted Today?

- **SUID/SGID**
  - No authentication required
  - Entire application runs with privilege
- **su**
  - Root password required to authenticate
  - Entire shell and all its child processes run with privilege
- **sudo**
  - User's password required to authenticate
  - Entire application runs with privilege
- **PolicyKit**
  - User's password required to authenticate (at login time)
  - Only certain, requested actions of application run with privilege

### Methods of Granting Privilege

**PolicyKit**

With PolicyKit, the user simply logs in as them self and executes the application. Depending on what action they try to perform with the application determines whether they can actually execute the application. The application only runs (with elevated privileges) for the actions that have been allowed. Delegation of privilege requires either the explicit granting of privilege on a per action basis or root authentication when the action is being attempted.
Introduction to PolicyKit

PolicyKit – Mechanism vs Policy

- Programs are split into 2 parts (separate processes):
  - Mechanism – Runs with privilege
  - Policy agent – Runs unprivileged

- The 2 parts use the system message bus (D-bus) to communicate

Notes:
Introduction to PolicyKit

The entities that the Mechanism works with:

- **Subject** - The “who” requesting the action
- **Object** - The “what” that is being acted upon
  
  Example: libvirt, device file, network connection to specific destination, power management, etc.

- **Action** - The “how” (what is requested to be done)
  
  Example: mounting a block device, establishing a dial-up connection, suspending the system, etc.
Introduction to PolicyKit

PolicyKit Authorization Process - Step 1

- A User requests that a Policy Agent perform an Action on a Mechanism (i.e. Launch a vm via libvirt)

Notes:
Introduction to PolicyKit

PolicyKit Authorization Process - Step 2

- The **Mechanism** uses **libpolkit** to check the **Authorization Database** to see if the action is **authorized** (are you root?)

The authorization check at this point is simply, are you root or have you been granted root privileges for this action?
Introduction to PolicyKit

PolicyKit Authorization Process - Step 3

- The **Mechanism** returns a denial to the **Policy Agent** because it is not privileged and therefore not authorized to perform the action (you're not root!)

```
User  --------> Policy Agent
          
User Session (unprivileged)

Action (virsh start)
Not Privileged (not root)

Mechanism
libpolkit

Authentication Agent
libpolkit-grant

Authorization Database

System Context (privileged)
```

Notes:
Introduction to PolicyKit

PolicyKit Authorization Process - Step 4

- If authorization has not already been granted, the Authentication Agent asks for root authentication
- On successful root authentication, a grant is given
  - Grant can be just this once, for entire session, or permanently depending on what is chosen

If there is not already an explicit grant of authorization, one can be placed into the PolicyKit database if the user requesting the action can provide the root password. If they can provide the root password, they are asked for the duration that the grant should be stored in the PolicyKit database. The duration could be: only for this action, for the entire session (until the user logs out), or permanently. If permanently is selected, a grant of authorization is stored in the PolicyKit database that same as if an explicit grant had been give previously.
Introduction to PolicyKit

PolicyKit Authorization Process - Step 5

- The Mechanism reads the granted authorization from the Authorization Database and allows the Agent to perform the Action (i.e. VM is launched)

Notes:
# Introduction to PolicyKit

## Important Files

<table>
<thead>
<tr>
<th>file</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/usr/lib/libpolkit.so</code></td>
<td>Main PolicyKit library</td>
</tr>
<tr>
<td><code>/etc/PolicyKit/PolicyKit.conf</code></td>
<td>Base PolicyKit configuration file</td>
</tr>
<tr>
<td><code>/var/lib/PolicyKit/</code></td>
<td>Contains user authorizations files</td>
</tr>
<tr>
<td><code>/var/lib/PolicyKit-public/</code></td>
<td>Contains default authorizations files</td>
</tr>
<tr>
<td><code>/usr/share/PolicyKit/policy/</code></td>
<td>Contains .policy files that define “Objects” that can be acted upon and how then can be acted upon</td>
</tr>
</tbody>
</table>

Notes:
# Introduction to PolicyKit

## PolicyKit Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>polkit-auth</td>
<td>CLI command to manage authorizations</td>
</tr>
<tr>
<td>polkit-gnome-authorization</td>
<td>GUI front end to polkit-auth</td>
</tr>
<tr>
<td>polkit-action</td>
<td>CLI command to manage actions that are registered on the system</td>
</tr>
<tr>
<td>polkit-config-file-validate</td>
<td>utility used to verify that a PolicyKit config file is valid before it is deployed</td>
</tr>
<tr>
<td>polkit-policy-file-validate</td>
<td>utility used to verify that .policy files are valid</td>
</tr>
</tbody>
</table>

Notes:
Introduction to PolicyKit

GNOME PolicyKit Authorizations utility

Notes:
Introduction to PolicyKit

 GNOME PolicyKit Authorizations - Actions

**Object(s)**

**Action(s)**

**Explicit Authorizations**

This list displays authorizations that are either obtained through authentication or specifically given to the entity in question. Blocked authorizations are marked with a STOP sign.

**Implicit Authorizations**

Implicit authorizations are authorizations automatically given to users based on certain criteria such as if they are on the local console.

- **Anyone:** Admin Authentication (keep indefinitely)
- **Console:** Admin Authentication (keep indefinitely)
- **Active Console:** Admin Authentication (keep indefinitely)

**Explicitly Granted Authorization**

**Authorities Based on Privilege/Policy**

**Notes:**
Delegate Virtualization Administration with PolicyKit

/etc/libvirt/libvirtd.conf

# Authentication.
#
# - none: do not perform auth checks. If you can connect to the
#   socket you are allowed. This is suitable if there are
#   restrictions on connecting to the socket (e.g. UNIX
#   socket permissions), or if there is a lower layer in
#   the network providing auth (e.g. TLS/SSL certificates)
#
# - sasl: use SASL infrastructure. The actual auth scheme is then
#   controlled from /etc/sasl2/libvirt.conf. For the TCP
#   socket only GSSAPI & DIGEST-MD5 mechanisms will be used.
#   For non-TCP or TLS sockets, any scheme is allowed.
#
# - polkit: use PolicyKit to authenticate. This is only suitable
#   for use on the UNIX sockets. The default policy will
#   require a user to supply their own password to gain
#   full read/write access (like sudo like), while anyone
#   is allowed read/only access.
#
# Set an authentication scheme for UNIX read-only sockets
# By default socket permissions allow anyone to connect
# To restrict monitoring of domains you may wish to enable
# an authentication mechanism here
#auth_unix_ro = "none"

# Set an authentication scheme for UNIX read-write sockets
# By default socket permissions only allow root. If PolicyKit
# support was compiled into libvirt, the default will be to
# use 'polkit' auth.

# If the unix_sock_rw_perms are changed you may wish to enable
# an authentication mechanism here
#auth_unix_rw = "none"
LAB 9-1: Use PolicyKit to Delegate VM Administration

**Summary:** In this exercise, you use PolicyKit to “authorize” a non-root user to use libvirt based VM management utilities.

**Special Instructions:**

**Duration:** 10 min.
Delegate Virtualization Administration with UNIX Group Membership
Delegate Virtualization Administration with UNIX Group Membership

UNIX Socket: Access

- Access to the UNIX socket can be limited to members of a UNIX group
- Can be combined with PolicyKit or SASL for authentication

Notes:
Delegate Virtualization Administration with UNIX Group Membership

/etc/libvirt/libvirtd.conf

```
# This is restricted to `root` by default.
# Set the UNIX domain socket group ownership. This can be used to
# allow a 'trusted' set of users access to management capabilities
# without becoming root.
# Set the UNIX socket permissions for the VDO socket. This is used
# for monitoring VM status only
# Default allows any user. If setting group ownership may want to
# restrict this to:
unix_sock_group = "libvirt"
unix_sock_ro_perms = "0777"
unix_sock_rw_perms = "0777"
unix_sock_dir = "/var/run/libvirt"
```

Notes:
LAB 9-2: Use UNIX Group Membership to Delegate VM Administration

Summary: In this exercise, you use UNIX group membership to “authorize” a non-root user to use libvirt based VM management utilities.

Special Instructions:

Duration: 10 min.
Delegate Virtualization Administration with SASL
Delegate Virtualization Administration with SASL

UNIX Socket: Auth

- Both Read Only and Read Write access to Libvirt can be restricted via SASL
- Can be combined with Delegation via UNIX group
- Allowed users are added to SASL database stored: `/etc/libvirt/passwd.db`

Notes:
Delegate Virtualization Administration with SASL

/etc/libvirt/libvirtd.conf

Notes:
Delegate Virtualization Administration with SASL

/etc/sasl2/libvirt.conf

```
# Before you can use GSSAPI, you need a service principle on the
# KDC server for Libvirt, and that to be exported to the keytab
# file listed below
#mech_list: gssapi
#
# You can also list many mechanisms at once, then the user can choose
# by adding 'auth=sasl.gssapi' to their libvirt URI, eg
# gssapi://hostname/system/auth=sasl.gssapi
#mech_list: digest-md5 gssapi
#
# MIT kerberos ignores this option & needs KRBS_KNAME env var.
# May be useful for other non-Linux OS though....
keytab: /etc/libvirt/krb5.tab

# If using digest-md5 for username/password, then this is the file
# containing the passwords. Use 'saslpasswd2 -a libvirt [username]'
# to add entries, and 'saslFillusers2 -a libvirt' to browse it
sasldb_path: /etc/libvirt/passwd.db

*/etc/sasl2/libvirt.conf* 2ML, H81C 1,1 All
```

Notes:
LAB 9-3: Use SASL+Digest-MD5 to Delegate VM Administration

Summary: In this exercise, you use SASL+Digest-MD5 to “authenticate” a non-root user to use libvirt based VM management utilities.

Special Instructions:

Duration: 10 min.

Lab Notes:
Objectives

- Pause and Save VMs
- Boot VMs Automatically
Pause and Save VMs
Pause and Unpause VMs

Pause/Unpause (Libvirt aka: suspend/resume)

- Pausing a Domain leaves the VM resident in memory but stops allocating it CPU cycles
  - Pausing VMs is supported for both paravirtual and full-virtual domains
- Unpausing a Domain starts allocating it CPU cycles again

Notes:
Manage VCPU Allocation

The virsh Command

- **Suspend/Resume**

  ```
  virsh suspend VM_NAME
  ```

  - Pause a VM leaving it resident in memory

  ```
  virsh resume VM_NAME
  ```

  - Unpause a paused VM

Notes:
Save and Restore VMs

Saving / Restoring

- Saving a managed or unmanaged VM pauses it, dumps the memory image to disk, and then removes it from memory
  - Saving requires you to specify the path to the “checkpoint” file in which to store the memory image
- Restoring a saved managed or unmanaged VM creates a new Domain container, loads the saved memory image into it, and then unpauses the Domain
  - Restoring does not remove the “checkpoint” file automatically
Suspend and Resume VMs

The virsh Command

• Save/Restore

virsh save VM_NAME SAVE_IMAGE

-Save a VM by specifying where the save image file is to be stored

virsh restore SAVE_IMAGE

-Restore a VM by specifying the save image file

Notes:
Suspend and Resume VMs

The virsh Command

- **Managedsave/Start**

  - `virsh managedsave VM_NAME`
    - Save a VM with libvirt deciding where to put the save image file
    - Flag a managed VM as having been saved for the virsh start command

  - `virsh start VM_NAME`
    - Resume a VM that has been flagged as having been saved

  - `virsh managedsave-remove VM_NAME`
    - Remove a managedsave image file
    - Unflag a VM from being saved

Notes:
LAB 10-2: Pause and Save KVM VMs

Summary: In this exercise you will suspend/resume, save/restore and managedsave/start a running KVM VM.

Special Instructions:

VM_NAME=sles11sp1-kvm
VM_DIR=/vmstore/sles11sp1-kvm/

Duration: 10 min.

Lab Notes:
Boot VMs Automatically
Boot VMs Automatically

**KVM**

- **libvirt-guests**
  - Both unmanaged (partial) and managed domains
- **libvirt**
  - Managed domains only

Notes:
Boot Xen VMs Automatically - libvirt-guests

- The libvirt-guests script is used to automatically start and stop VMs when the vHost is started/stopped
- Works for KVM and libxenlight Xen VMs
- Configuration of the libvirt-guests script is done by editing variables listed in its configuration file
  - Config file: /etc/sysconfig/libvirt-guests

* Recommended method for KVM and Xen with libxenlight

Notes:
Boot Xen VMs Automatically - libvirt-guests
/etc/sysconfig/libvirt-guests

- **Important values:**
  - **ON_BOOT**
    - start = all guests that were running on shutdown get restarted/resumed
    - ignore = no guests that were running on shutdown get restarted/resumed (unless flagged for libvirt autostart anyway)
  - **START_DELAY**
    - This defines the amount of time in seconds to wait before each guest start
  - **ON_SHUTDOWN**
    - suspend =
    - shutdown =
  - **SHUTDOWN_TIMEOUT**
    - This defines the amount of time in seconds to wait for a guest to stop before moving on

Notes:
Boot Libvirt VMs Automatically - Managed with Libvirt

*KVM* & *Xen (w/ libxenlight)*

- The automatic starting and stopping/suspending of domains can be performed by the *Libvirt* daemon
- Command:
  
  ```bash
  virsh autostart VM_NAME
  ```

  » Creates a symlink/copy of the VM's xml config in:
  
  ```bash
  /etc/libvirt/qemu/autostart/
  ```

  ```bash
  virsh autostart --disable VM_NAME
  ```

  » Disables autostart of a VM

* Recommended method for KVM and Xen with libxenlight

Notes:
LAB 10-4: Automatically Boot KVM VMs with libvirt-guests

Summary: In this exercise you will configure the vHost to automatically start and stop KVM VMs when it boots and shuts down.

Special Instructions:

VM_NAME=sles11sp1-kvm
VM_CONFIG=/vmstore/sles11sp1-kvm/sles11sp1-kvm.xml

Duration: 10 min.
Objectives

- Pause and Save VMs
- Boot VMs Automatically
Serial Port and Console Devices
User Input/Output Devices – Serial Port

- **KVM Virtual Machines**
  - Serial port(s) can be passed through to a KVM VM
    - Typically used for debugging purposes
  - XML config file syntax (physical device):

```xml
<serial type='dev'>
  <source path='/dev/ttyS0' />
  <target port='0' />
</serial>
<console type='dev'>
  <source path='/dev/ttyS0' />
  <target port='0' />
</console>
```

Serial ports in the vHost can be mapped into virtual serial ports in the VMs. This is typically only done in cases where debug information is needed to be captured from a serial port.
User Input/Output Devices – Console

- **KVM & Xen (HVM)** - Virtual Machines

  - A serial device must be configured as the console
    - In a Linux VM, it must be enabled in `/etc/inittab` and `/etc/securetty`
    - `inittab`: `S0:1234:respawn:/sbin/agetty -L 9600 ttyS0 vt102`
  
  - XML Config File Syntax:
    ```xml
    <serial type='pty'>
      <source path='/dev/pts/2' />
      <target port='0' />
      <alias name='serial0' />
    </serial>
    <console type='pty'>
      <source path='/dev/pts/2' />
      <target port='0' />
      <alias name='serial0' />
    </console>
    ```

  Only paravirtual machines have access to the virtual console. Due to limitations in the qemu-dm, the console device is not exposed to the OSes running in full-virtual machines.

  The virtual console should be treated in most cases like a serial console. When first attaching to the console, you must press Enter to get the console to display output to you.
User Input/Output Devices – Console

**KVM**

- **Connect to the Console**
  - Connect to the console of a running VM
    - `virsh console VM_Name or VM_ID`
  - Detach from the console
    - `ctrl+]` (control + right square bracket)

---

Only paravirtual machines have access to the virtual console. Due to limitations in the qemu-dm, the console device is not exposed to the OSes running in full-virtual machines. The virtual console should be treated in most cases like a serial console. When first attaching to the console, you must press Enter to get the console to display output to you.
LAB 2-1: Enable a Serial Console in a Full-virtual Machine

Summary: In this exercise, you enable a serial console in a full-virtual machine

Special Instructions
Use the following values in the exercise:

- `VM_NAME=sles11sp1-kvm`
- `VM_CONFIG=sles11sp1-kvm.xml`
- `VM_DIR=/vmstore/sles11sp1-kvm/`

Duration: 10 min.
Watchdog Devices

- **KVM & Xen** (HVM) - Virtual Machines
  - A watchdog device can be presented to a VM
  
  > Supported models:
  >   » i6300esb - emulates PCI Intel 6300ESB (recommended)
  >   » ib700 - emulates ISA iBase IB700
  
  > Supported actions:
  >   » reset = forceful reset of VM (default)
  >   » shutdown = graceful shutdown of VM
  >   » poweroff = forceful power off of VM
  >   » pause = pause the VM
  >   » none = do nothing
  >   » dump = automatically dump the VM

- XML Config File Syntax:

  ```xml
  <device>
    <watchdog model='i6300esb' action='poweroff' />
  </device>
  ```

Notes:
Manage Virtual Networks with libvirt
Manage Virtual Networks with libvirt

Libvirt virtual network management

- Libvirt has the ability to dynamically create virtual networks

- Virtual networks created by libvirt are not persistent unless “defined” and set to “autostart”
  - The xml configuration for defined (persistent) virtual networks is stored in `/etc/libvirt/qemu/networks/`
  - The xml configuration for “autostarted” virtual networks is stored in `/etc/libvirt/qemu/networks/autostart/`

- Libvirt can create basic, NATed, and routed virtual networks

Notes:
Manage Virtual Networks with Virt-Manager

Notes:
Manage Virtual Networks with Virt-Manager

Choosing an IPv4 address space

You will need to choose an IPv4 address space for the virtual network:

Network: 192.168.11.0/24

Hint: The network should be chosen from one of the IPv4 private address ranges: e.g., 10.0.0.0/8, 172.16.0.0/12, or 192.168.0.0/16.

- Netmask: 255.255.255.0
- Broadcast: 192.168.11.255
- Gateway: 192.168.11.1
- Size: 256 addresses
- Type: Private

Notes:
Manage Virtual Networks with Virt-Manager

Selecting the DHCP range

Please choose the range of addresses the DHCP server will allocate to virtual machines attached to the virtual network.

Enable DHCP: ✔

Start: 192.168.11.128
End: 192.168.11.254

Tip: Unless you wish to reserve some addresses to allow static network configuration in virtual machines, these parameters can be left with their default values.

Connecting to physical network

Please indicate whether this virtual network should be connected to the physical network:

- Isolated virtual network
- Forwarding to physical network

Destination: Any physical device
Mode: NAT

Notes:
Manage Virtual Networks with virsh

Display Network Configuration

**Command**

```
virsh net-list [--all | --inactive]
```
Displays a list of virtual networks

```
virsh net-dumpxml VNET_NAME
```
Displays the xml configuration for a virtual network

**Notes:**
Manage Virtual Networks with virsh

Create/Define Virtual Networks

**Command**

- `virsh net-create XML_FILE`
  Creates a new virtual network from an xml definition file

- `virsh net-define XML_FILE`
  Defines but doesn't activate a new virtual network from an xml file

- `virsh net-start VNET_NAME`
  Starts a defined but not currently active virtual network

- `virsh net-autostart VNET_NAME`
  Sets a defined virtual network to start

**Notes:**
Manage Virtual Networks with `virsh`

Remove Virtual Networks

**Command**

```bash
virsh net-undefine XML_FILE
```

Undefines and inactive virtual network

```bash
virsh net-destroy VNET_NAME
```

Removes a virtual network

**Notes:**
Libvirt Virtual Network Definitions

Basic Bridge:

```xml
<network>
  <name>br0</name>
  <uuid>37d9af4c-464a-47bb-923d-e73edc81122f</uuid>
  <bridge name='' stp='off' forwardDelay='0' />
  <ip address='192.168.1.1' netmask='255.255.255.0' />
</network>
```

NATed Bridge:

```xml
<network>
  <name>nat0</name>
  <uuid>37d9af4c-464a-47bb-923d-e73edc81122f</uuid>
  <bridge name='' stp='off' forwardDelay='0' />
  <forward dev='eth0' mode='nat' />
  <ip address='192.168.2.1' netmask='255.255.255.0' />
</network>
```

Notes:
LAB 8-8: Configure a Virtual Network with libvirt

Summary: In this exercise, you use libvirt to configure a virtual network.

Special Instructions
None

Duration: 15 min.

Lab Notes:
Snapshot Virtual Disk Images
Snapshot Virtual Disk Images

QCOW2 and VHD files can be snapshotted using the `vm-snapshot-disk` command

Syntax:

Create Snapshot
`vm-snapshot-disk create disk=<disk_file>`

Create a new snapshot branch
`vm-snapshot-disk branch disk=<disk_file> snapname=<snap_name>`

Revert to a snapshot
`vm-snapshot-disk revert disk=<disk_file> snapname=<snap_name>`

Remove all snapshots of a disk
`vm-snapshot-disk remove disk=<disk_file>`

Notes:
LAB 6-9: Snaphot a QCOW2 Virtual Disk

Summary: In this exercise, you snapshot a qcow2 disk image.

Special Instructions
Use the following values in the exercise:

VM_NAME
VM_DIR
DISK_IMAGE

Duration: 10 min.

Lab Notes: