

# IBM PowerKVM Configuration and Use

Murilo Opsfelder Araújo

Breno Leitao

Stephen Lutz

José Ricardo Ziviani



 **Cloud**

**Power Systems**





International Technical Support Organization

**IBM PowerKVM: Configuration and Use**

March 2016

**Note:** Before using this information and the product it supports, read the information in “Notices” on page xvii.

## **Second Edition (March 2016)**

This edition applies to Version 3, Release 1, Modification 0 of IBM PowerKVM (product number 5765-KV3).

© Copyright International Business Machines Corporation 2014, 2016. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.



# Contents

<b>Figures</b> .....	vii
<b>Tables</b> .....	xi
<b>Examples</b> .....	xiii
<b>Notices</b> .....	xvii
Trademarks .....	xviii
<b>IBM Redbooks promotions</b> .....	xix
<b>Preface</b> .....	xxi
Authors .....	xxi
Now you can become a published author, too! .....	xxii
Comments welcome .....	xxiii
Stay connected to IBM Redbooks .....	xxiii
<b>Summary of changes</b> .....	xxv
March 2016, Second Edition .....	xxv
<b>Chapter 1. Introduction</b> .....	1
1.1 IBM Power Systems .....	2
1.1.1 POWER8 processors .....	2
1.1.2 IBM scale-out servers .....	3
1.1.3 Power virtualization .....	7
1.1.4 Simultaneous multithreading .....	8
1.1.5 Memory architecture .....	8
1.1.6 Micro-Threading .....	9
1.1.7 RAS features .....	10
1.2 Virtualization .....	10
1.2.1 PowerKVM versions .....	11
1.2.2 PowerKVM Version 3.1 considerations .....	12
1.2.3 Where to download PowerKVM .....	12
1.3 Software stack .....	13
1.3.1 QEMU .....	13
1.3.2 KVM .....	14
1.3.3 Open Power Abstraction Layer .....	14
1.3.4 Guest operating system .....	15
1.3.5 Libvirt software .....	15
1.3.6 Virsh interface .....	16
1.3.7 Intelligent Platform Management Interface .....	17
1.3.8 Petitboot .....	17
1.3.9 Kimchi .....	18
1.3.10 Slimline Open Firmware .....	19
1.3.11 Virtio drivers .....	20
1.3.12 RAS stack .....	21
1.4 Docker .....	21
1.4.1 Architecture .....	24
1.4.2 Docker hub .....	25
1.4.3 Docker file .....	26

1.5 Comparisons of PowerVM and PowerKVM .....	26
1.5.1 PowerVM and PowerKVM features. ....	27
1.5.2 PowerKVM Version 2.1 and Version 3.1. ....	27
1.6 Terminology .....	29
<b>Chapter 2. Host installation and configuration</b> .....	31
2.1 Host installation. ....	32
2.2 Boot over network .....	43
2.2.1 Retrieve configuration from URL. ....	44
2.2.2 Boot LiveDVD over network .....	45
2.2.3 Automated boot over DHCP .....	46
2.2.4 Automated installation. ....	48
2.3 Install over existing IBM PowerKVM and host migration. ....	52
2.4 Configuration. ....	55
2.4.1 Console configuration for Scale-out Power Systems .....	58
2.4.2 Console configuration for Power LC systems .....	59
<b>Chapter 3. Managing hosts and guests from a Web interface</b> .....	61
3.1 Kimchi .....	62
3.1.1 Accessing Kimchi .....	62
3.1.2 Getting help. ....	63
3.2 Host tab. ....	63
3.3 Storage pool .....	67
3.3.1 Local directory. ....	68
3.3.2 NFS. ....	71
3.3.3 iSCSI. ....	72
3.3.4 Logical volume .....	74
3.4 Network. ....	75
3.4.1 NAT network .....	77
3.4.2 Bridged network .....	78
3.4.3 Isolated network .....	79
3.5 Templates .....	79
3.5.1 Create a new template .....	80
3.5.2 Edit an existing template. ....	81
3.5.3 Create a template from an existing disk image. ....	84
3.6 Guests. ....	85
3.6.1 Create a new guest. ....	85
3.6.2 Guest management .....	86
3.7 Accessing a graphical interface .....	95
3.7.1 noVNC .....	95
3.7.2 VNC .....	95
3.7.3 Custom keyboard layouts in Kimchi noVNC .....	96
3.8 Ginger .....	96
<b>Chapter 4. Managing guests from the command-line interface</b> .....	105
4.1 virsh console .....	106
4.1.1 virsh vncdisplay. ....	107
4.2 Managing storage pools .....	107
4.2.1 Create new storage pools. ....	107
4.2.2 Query available storage pools .....	110
4.2.3 List available volumes. ....	111
4.2.4 Create a new volume .....	111
4.2.5 Delete or wipe a volume .....	112
4.2.6 Snapshots. ....	112

4.3	Manage guest networks	114
4.3.1	Query guest networks	114
4.3.2	Create a guest network	115
4.4	Managing guests	117
4.4.1	Create a new guest	117
4.4.2	List guests	118
4.4.3	Start or stop a guest	119
4.4.4	Suspending and resuming	120
4.4.5	Delete a guest	120
4.4.6	Connect to a guest	121
4.4.7	Edit a guest	121
4.4.8	Add new storage to an existing guest	122
4.4.9	Add a new network to an existing guest	123
4.4.10	PCI I/O pass-through	124
4.4.11	CPU Hotplug	127
4.4.12	Memory Hotplug	128
4.4.13	Clone a guest	128
4.4.14	Migration	128
<b>Chapter 5.</b>	<b>Processor and memory virtualization</b>	<b>129</b>
5.1	CPU virtualization	130
5.1.1	Types of virtualization	130
5.2	CPU overcommitment	131
5.3	CPU configuration	132
5.3.1	CPU compatibility mode	132
5.3.2	Simultaneous multithreading	133
5.3.3	Micro-Threading	136
5.3.4	Configuring NUMA	141
5.3.5	CPU pinning	142
5.3.6	CPU shares	144
5.4	CPU Hotplug	145
5.4.1	CPU Hotplug with a NUMA configuration	148
5.4.2	Considerations for CPU Hotplug	150
5.5	Memory	150
5.5.1	Memory allocation	151
5.5.2	Memory ballooning	151
5.5.3	Kernel SamePage Merging	153
5.5.4	Huge pages	155
5.5.5	Restrict NUMA memory allocation	156
5.6	Memory Hotplug	157
<b>Chapter 6.</b>	<b>I/O virtualization</b>	<b>163</b>
6.1	Types of virtualization	164
6.1.1	PowerKVM supported devices	164
6.1.2	PCI I/O pass-through	164
6.2	Network virtualization	164
6.2.1	User mode networking	165
6.2.2	Network address translation networking	165
6.2.3	Bridged networking	165
6.2.4	Open vSwitch	166
6.3	Storage pools	167
6.3.1	Storage volume	167
6.3.2	Block device pools	167

6.3.3 File-backed pools . . . . .	168
6.4 I/O pass-through . . . . .	170
6.4.1 SCSI pass-through . . . . .	170
6.4.2 USB pass-through . . . . .	171
6.4.3 PCI pass-through to a virtual machine . . . . .	172
6.4.4 I/O limits . . . . .	175
6.5 N_Port ID Virtualization (NPIV) . . . . .	176
6.6 Using multipath disks . . . . .	182
6.6.1 Multipath disk handling . . . . .	182
6.6.2 Direct mapped multipath disks . . . . .	185
6.6.3 Multipath disks in a storage pool . . . . .	187
6.7 Hot plug . . . . .	189
6.7.1 Adding a new vSCSI adapter . . . . .	191
<b>Chapter 7. Advanced topics . . . . .</b>	<b>193</b>
7.1 Install PowerKVM on a hardware RAID . . . . .	194
7.2 Guest migration . . . . .	198
7.2.1 Offline migration . . . . .	198
7.2.2 Online migration . . . . .	199
7.2.3 Live migration . . . . .	199
7.3 Booting PowerKVM from Petitboot shell . . . . .	200
7.4 Security . . . . .	201
7.4.1 SELinux . . . . .	202
7.4.2 System updates . . . . .	203
7.5 Cloud management . . . . .	204
7.5.1 IBM PowerVC . . . . .	204
7.5.2 IBM Cloud Manager with OpenStack . . . . .	210
7.5.3 OpenStack controller services . . . . .	211
7.6 Docker usage . . . . .	214
7.6.1 Docker installation . . . . .	214
7.6.2 Image management . . . . .	217
7.6.3 Container management . . . . .	218
7.6.4 Uploading your image to Docker hub . . . . .	222
7.6.5 Creating image from scratch . . . . .	223
<b>Chapter 8. PowerKVM Development Kit . . . . .</b>	<b>227</b>
8.1 Introduction . . . . .	228
8.1.1 Libvirt API . . . . .	228
8.2 Installation . . . . .	229
8.3 Architecture . . . . .	231
8.4 Initial example . . . . .	233
8.4.1 Using Python binding . . . . .	234
8.4.2 Using the API in C . . . . .	235
8.5 Query memory and CPU utilization example . . . . .	237
8.6 Query guest network information . . . . .	240
<b>Related publications . . . . .</b>	<b>247</b>
IBM Redbooks . . . . .	247
Online resources . . . . .	247
Help from IBM . . . . .	249

# Figures

1-1	12-core IBM POWER8 architecture	2
1-2	Front view of the Power S812L	3
1-3	Front view of the Power S822L	4
1-4	S821L and S822L rear view	4
1-5	S824L front view	5
1-6	S824L rear view	5
1-7	NVIDIA Tesla adapter	6
1-8	Front view of the Power S822LC	6
1-9	Front view of the Power S812LC	7
1-10	Local and remote memory access in a NUMA architecture	9
1-11	IBM POWER8 core with Micro-Threading enabled	10
1-12	Guests and hypervisor	11
1-13	QEMU monitor window example	13
1-14	Virsh and libvirt architecture	16
1-15	Petitboot window	17
1-16	Kimchi home panel	18
1-17	SLOF during VM boot	19
1-18	Virtio architecture for network stack	20
1-19	Namespaces example	22
1-20	Change root feature	23
1-21	Containers and virtual images on a POWER server	24
1-22	Container and guest OS stack	25
1-23	Build and run process	26
2-1	Petitboot menu	33
2-2	Petitboot system configuration	34
2-3	Language selection	34
2-4	Welcome to IBM PowerKVM	35
2-5	Select the target device to install PowerKVM	36
2-6	Change the name of the volume group and the size of the volumes if needed	36
2-7	Root password	37
2-8	Time zone selection	37
2-9	NTP configuration	38
2-10	Date and time configuration	38
2-11	Configure network	39
2-12	Network device configuration	39
2-13	DNS configuration	40
2-14	Installation summary	40
2-15	Device confirmation	41
2-16	Installer progress bar	41
2-17	Reboot the system	42
2-18	PowerKVM is automatically loaded from the installed device after reboot	42
2-19	Petitboot Retrieve config from URL entry	44
2-20	Petitboot Config Retrieval window	44
2-21	Petitboot configuration retrieved	45
2-22	Install over existing IBM PowerKVM option displayed in the installer menu	53
2-23	Select existing installation window	54
2-24	The ibm-configure-system tool	55
2-25	Reset password	56

2-26	Timezone selection . . . . .	56
2-27	Select network device . . . . .	57
2-28	Network device configuration . . . . .	57
2-29	DNS configuration change . . . . .	58
2-30	Select firmware configuration . . . . .	58
2-31	Changing the IPMI password . . . . .	59
3-1	Login panel . . . . .	63
3-2	Accessing help . . . . .	63
3-3	Host tab selected . . . . .	63
3-4	Host tab information . . . . .	64
3-5	Host system statistics . . . . .	65
3-6	Package update and repository manager . . . . .	65
3-7	Adding a new repository . . . . .	66
3-8	Generating a new SOS Report . . . . .	66
3-9	Generating a new SOS Report . . . . .	67
3-10	List of reports created . . . . .	67
3-11	Storage tab . . . . .	68
3-12	Local directory storage pool . . . . .	69
3-13	Activating a new storage pool . . . . .	70
3-14	NFS storage pool . . . . .	71
3-15	iSCSI storage pool . . . . .	72
3-16	iSCSI volumes list . . . . .	73
3-17	Creating a template using an iSCSI volume . . . . .	73
3-18	LVM storage pool . . . . .	74
3-19	LVM storage pool . . . . .	75
3-20	Network tab . . . . .	75
3-21	Starting a network . . . . .	76
3-22	Creating a new NAT network . . . . .	77
3-23	Creating a bridged network . . . . .	78
3-24	Creating an isolated network . . . . .	79
3-25	Template tab . . . . .	80
3-26	Selecting the Local ISO Image . . . . .	80
3-27	Selecting the operating system . . . . .	80
3-28	Editing a template . . . . .	81
3-29	Template editor . . . . .	81
3-30	Storage tab in the template editor . . . . .	82
3-31	Network tab in the template editor . . . . .	83
3-32	Processor tab in template editor . . . . .	83
3-33	Editing CPU topology in template editor . . . . .	84
3-34	Selecting the local image file . . . . .	84
3-35	Image disk file path . . . . .	85
3-36	The guest tab . . . . .	85
3-37	Creating a new virtual machine . . . . .	86
3-38	Guest control buttons for stopped guest . . . . .	87
3-39	Edit a stopped guest . . . . .	87
3-40	Storage tab interface . . . . .	88
3-41	Attach an existing volume . . . . .	88
3-42	Interface tab . . . . .	89
3-43	Adding a network . . . . .	89
3-44	Permission tab . . . . .	90
3-45	Host PCI device tab . . . . .	90
3-46	Snapshot tab . . . . .	91
3-47	Guest control buttons for a running guest . . . . .	91

3-48	Attach a storage to a running guest . . . . .	92
3-49	Adding a new network interface . . . . .	93
3-50	List of PCI devices . . . . .	94
3-51	List of PCI devices . . . . .	95
3-52	Administration tab . . . . .	97
3-53	Firmware update tool . . . . .	97
3-54	Selecting an image from IBM Fix Central . . . . .	98
3-55	Ginger configuration backup tool . . . . .	98
3-56	Creating a custom backup . . . . .	99
3-57	Removing old backups . . . . .	99
3-58	Network configuration tool . . . . .	100
3-59	Power Options tool . . . . .	100
3-60	Changing a profile . . . . .	100
3-61	Listing SAN adapters connected to the system . . . . .	101
3-62	Sensor monitor . . . . .	101
3-63	Starting SEP service . . . . .	101
3-64	Subscribing a new listener . . . . .	102
3-65	SEP configuration pane . . . . .	102
3-66	Listing existing users on the host . . . . .	102
3-67	Adding a new user . . . . .	103
4-1	Deleting a snapshot . . . . .	113
4-2	Interaction between host and guest during PCI hotplug . . . . .	125
5-1	CPU overcommitment scenario . . . . .	131
5-2	Example of a POWER8 core with four subcores and two threads each subcore . . . .	137
5-3	Four virtual machines running in a single core without Micro-Threading enabled . . .	137
5-4	Four virtual machines running in a single core with Micro-Threading enabled . . . .	138
5-5	KSM mapping when VM uses the same page . . . . .	153
6-1	Bridge architecture . . . . .	166
6-2	File-backed pool . . . . .	168
6-3	iSCSI-backed storage pool . . . . .	169
6-4	NFS-backed storage device . . . . .	169
6-5	Fibre Channel-backed storage device . . . . .	170
6-6	Virtual Fibre Channel adapters in Kimchi . . . . .	178
6-7	Typical multipath environment . . . . .	182
6-8	IBM Storwize V7000 storage view of LUNs attached to PowerKVM. . . . .	183
6-9	Added multipath disk in Kimchi . . . . .	187
7-1	iprconfig main window . . . . .	194
7-2	Create a disk array option . . . . .	195
7-3	Select disk adapter . . . . .	195
7-4	Select disk units . . . . .	196
7-5	Select RAID type . . . . .	196
7-6	Confirmation window for creating disk array . . . . .	197
7-7	Disk array status . . . . .	197
7-8	PowerVC interface for advanced virtualization management . . . . .	205
7-9	Adding a new host to PowerVC . . . . .	205
7-10	Hosts list on PowerVC . . . . .	206
7-11	Image upload on PowerVC . . . . .	207
7-12	Displaying images . . . . .	208
7-13	Deploy an image on PowerVC . . . . .	208
7-14	Attachment of an iSCSI disk in PowerVC . . . . .	209
7-15	IBM Cloud Manager with OpenStack Self Service portal . . . . .	211
8-1	virConnect structure . . . . .	233





# Tables

1-1	Recommended firmware version for PowerKVM	7
1-2	PowerKVM versions	11
1-3	Maximum amount of resources per virtual machine	12
1-4	Frequently used IPMI commands	17
1-5	Virtio drivers	20
1-6	Differences between containers and virtual machines	25
1-7	Comparison of IBM PowerKVM and IBM PowerVM features	27
1-8	PowerKVM releases comparison	27
1-9	Terminology comparing KVM and PowerVM	29
2-1	Netboot supported protocols	45
5-1	The relation between vCPU, cores, and threads on guest configuration	136
5-2	Threads per subcore	136
5-3	Supported values for dynamic_mt_modes	139
5-4	Required packages to support CPU and memory Hotplug	145
5-5	KSM options	155
7-1	Virtualization management systems	204



# Examples

2-1	Petitboot configuration file . . . . .	46
2-2	DHCP configuration sample . . . . .	46
2-3	Boot configuration sample . . . . .	47
2-4	TFTP configuration sample . . . . .	47
2-5	Petitboot configuration sample for unattended installation . . . . .	48
2-6	Kickstart sample for installing PowerKVM on a single disk . . . . .	49
2-7	Kickstart example using more than one disk for installation . . . . .	49
2-8	Install PowerKVM V3.1.0 on an existing PowerKVM V2.1.1 instance . . . . .	50
2-9	Configure an interface with DHCP . . . . .	50
2-10	Configure an interface with static IP address . . . . .	50
2-11	Kickstart file sample . . . . .	51
2-12	Convert libvirt configuration files using powerkvm-xml-tool . . . . .	54
3-1	Guest disks . . . . .	92
3-2	Guest disks . . . . .	92
3-3	Guest network interfaces . . . . .	93
3-4	Guest network interfaces . . . . .	93
3-5	Listing all devices in the guest . . . . .	94
3-6	Listing all devices in guest . . . . .	94
3-7	Listing all devices in guest . . . . .	95
3-8	Obtaining the VNC display number . . . . .	96
3-9	keymap setting in guest XML . . . . .	96
4-1	Connect remotely . . . . .	106
4-2	Working within virsh interactive shell . . . . .	106
4-3	Running virsh commands from system shell . . . . .	106
4-4	vncdisplay command . . . . .	107
4-5	Create a file-based pool . . . . .	107
4-6	Create block-based pools . . . . .	109
4-7	Query available storage pools . . . . .	110
4-8	Display pool information . . . . .	111
4-9	Display a verbose pool list . . . . .	111
4-10	Display volume list . . . . .	111
4-11	Create volume in storage pool . . . . .	111
4-12	Display the volume list . . . . .	112
4-13	Display verbose volume list . . . . .	112
4-14	Wipe volume . . . . .	112
4-15	Volume delete . . . . .	112
4-16	Working with snapshots . . . . .	113
4-17	Command to show the bridge interfaces on the system . . . . .	114
4-18	Display networks . . . . .	114
4-19	Display specific network . . . . .	114
4-20	NAT definition . . . . .	115
4-21	Bridged network . . . . .	116
4-22	Open vSwitch configuration . . . . .	116
4-23	Open vSwitch bridge interface . . . . .	117
4-24	vcpus options . . . . .	118
4-25	Disk options . . . . .	118
4-26	LUN mapping . . . . .	118
4-27	Network arguments . . . . .	118

4-28	virt-install example	118
4-29	List all guests	119
4-30	Start VM	119
4-31	Stop/halt VM	119
4-32	Suspending and resuming a guest	120
4-33	Deleting a guest	120
4-34	Console access	121
4-35	Editing guest configuration	121
4-36	Using another text editor	122
4-37	Adding new virtual storage	122
4-38	Storage live pass-through	122
4-39	Network interface pass-through	123
4-40	Network interface hotplug	123
4-41	Listing host PCI devices	124
4-42	Getting PCI device information	124
4-43	Attaching a multi-function PCI device	126
4-44	Guest displaying multifunction device	127
4-45	Reattaching multifunction device to the host	127
4-46	Creating a clone guest	128
5-1	Enable POWER7 compatibility mode	132
5-2	Guest in POWER7 compatibility mode	132
5-3	Enable POWER6 compatibility mode	133
5-4	SMT disabled on the hypervisor	133
5-5	Setting the number of threads per core	134
5-6	CPU information about a guest with SMT	134
5-7	Only 4-way dynamic Micro-Threading	139
5-8	Checking if Micro-Threading is enabled	140
5-9	Definition of a NUMA guest	141
5-10	Verification of a NUMA configuration inside the guest	141
5-11	CPU pinning without SMT	142
5-12	Verification on CPU pinning	143
5-13	CPU pinning with SMT	143
5-14	CPU pinning with subcores	144
5-15	CPU shares	144
5-16	Base definition of sockets, cores, and threads for CPU Hotplug	145
5-17	XML snippet for Hotplugging a socket	146
5-18	CPU Hotplug example	146
5-19	Change SMT mode and Hotplug another socket	147
5-20	Removal of one socket using CPU Hotplug	148
5-21	CPU Hotplug with a NUMA configuration	148
5-22	Unsupported configuration for CPU Hotplug	150
5-23	Memory allocation	151
5-24	Enable memory balloon on the guest	151
5-25	Decreasing the virtual machine memory to 1 GB	152
5-26	Output of memory available on the balloon	152
5-27	Changing the memory allocated to the virtual machine	152
5-28	Verify that the ksmtuned daemon is running	154
5-29	Enable KSM in PowerKVM	154
5-30	Number of pages shared in the hypervisor	154
5-31	Number of pages that are linked to a shared page	154
5-32	Setting huge pages on the host	155
5-33	Enabling huge pages on a guest	155
5-34	Error starting a guest with huge pages	156

5-35	Memory allocation to NUMA nodes before restricting it to one node . . . . .	156
5-36	NUMA node set. . . . .	156
5-37	Memory allocation to NUMA nodes after restricting it to one node . . . . .	157
5-38	XML snippet for a DIMM with 4 GB. . . . .	157
5-39	Example of how to increase the memory by using memory hotplug. . . . .	158
5-40	Persistent attachment of DIMMs. . . . .	159
5-41	Memory Hotplug snippet in a NUMA environment . . . . .	159
5-42	Memory Hotplug within a NUMA configuration . . . . .	159
5-43	Persistent Hotplug Memory DIMMs . . . . .	160
6-1	LUN pass-through . . . . .	170
6-2	lsusb command output . . . . .	171
6-3	USB XML description example based on vendor, product (IDs) pair . . . . .	171
6-4	USB XML description example based on bus, device pair . . . . .	171
6-5	lsusb command output from a virtual machine . . . . .	171
6-6	Attach the device to a VM. . . . .	172
6-7	Detaching a USB device from a guest . . . . .	172
6-8	lspci command output . . . . .	172
6-9	virsh nodeudev-list command output . . . . .	173
6-10	virsh nodeudev-dumpxml command output . . . . .	173
6-11	virsh nodeudev-detach command . . . . .	174
6-12	PCI adapter description . . . . .	174
6-13	lspci command output within the VM . . . . .	174
6-14	List of available interfaces. . . . .	175
6-15	Change disk properties . . . . .	175
6-16	iotune example of a disk element . . . . .	176
6-17	Discovery of Fibre Channel adapters . . . . .	176
6-18	Creating a virtual Fibre Channel adapter . . . . .	177
6-19	New disk at a virtual Fibre Channel adapter . . . . .	179
6-20	Creating a storage pool for a virtual Fibre Channel adapter . . . . .	180
6-21	Attachment of a disk from a storage pool on a virtual Fibre Channel adapter . . . . .	180
6-22	Example for multipath disk . . . . .	182
6-23	Changes in /etc/multipath.conf . . . . .	184
6-24	Output of multipath -ll after some changes . . . . .	184
6-25	Example of a path failure . . . . .	185
6-26	XML snippet for attachment of a multipath disk into a guest . . . . .	185
6-27	Mapped multipath disk inside the guest . . . . .	186
6-28	LUN mapped as path-through device . . . . .	186
6-29	Using multipath disks in a storage pool . . . . .	187
6-30	vSCSI disk hot plug. . . . .	189
6-31	vSCSI hot plug attachment to the guest . . . . .	189
6-32	Detach a disk from the guest . . . . .	190
6-33	Additional adapter definition . . . . .	191
7-1	Offline migration . . . . .	199
7-2	Online migration . . . . .	199
7-3	Live migration . . . . .	200
7-4	Live migration with timeout . . . . .	200
7-5	boot.sh . . . . .	201
7-6	Sample SELinux configuration file . . . . .	202
7-7	Default repository configuration file. . . . .	203
7-8	Simple open vSwitch environment configured by PowerVC. . . . .	206
7-9	Listing hosts on Nova . . . . .	212
7-10	Deploying an image using a nova command line . . . . .	213
7-11	Installing Docker package on PowerKVM. . . . .	214

7-12	docker info output . . . . .	216
7-13	Docker service access error . . . . .	216
7-14	Docker failing due to shared library not found . . . . .	217
7-15	Searching for images . . . . .	217
7-16	Downloading a remote Docker image . . . . .	218
7-17	Listing container images . . . . .	218
7-18	Starting a container . . . . .	219
7-19	Listing active containers . . . . .	219
7-20	Listing all containers . . . . .	219
7-21	Creating a Docker container based on an image . . . . .	220
7-22	Getting the Docker console . . . . .	220
7-23	Docker image changes . . . . .	221
7-24	Committing a file system change to a new image . . . . .	221
7-25	Renaming a Docker image . . . . .	221
7-26	Login information about Docker hub . . . . .	222
7-27	Uploading an image to Docker hub . . . . .	222
7-28	Finding the image previously uploaded . . . . .	223
7-29	Importing Ubuntu Core from the web . . . . .	223
7-30	Renaming an image and starting it . . . . .	223
7-31	Debootstrapping Debian in a local directory . . . . .	224
7-32	Importing from a local tgz file . . . . .	225
8-1	Assuring the online repository is configured . . . . .	229
8-2	Development packages . . . . .	230
8-3	Repository file for development kit installation from ISO . . . . .	231
8-4	Functions used to open a connection . . . . .	231
8-5	SASL access . . . . .	232
8-6	First example in Python . . . . .	234
8-7	Running the first example . . . . .	235
8-8	Connecting to a localhost hypervisor . . . . .	235
8-9	Return a domain ID for a domain name . . . . .	236
8-10	First example main body . . . . .	236
8-11	Get the memory information for the active guests . . . . .	237
8-12	Second development example . . . . .	238
8-13	Output of the second example . . . . .	240
8-14	Initial program implementation . . . . .	240
8-15	get_mac implementation . . . . .	241
8-16	print_ip_per_mac implementation . . . . .	242
8-17	Implementing the main function . . . . .	243
8-18	Showing the final results . . . . .	244
8-19	Python implementation . . . . .	244
8-20	Program results . . . . .	245

# Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US*

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

## COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

# Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks or registered trademarks of International Business Machines Corporation, and might also be trademarks or registered trademarks in other countries.

Active Memory™  
AIX®  
IBM®  
IBM SmartCloud®  
IBM z Systems™  
Micro-Partitioning®

POWER®  
Power Architecture®  
Power Systems™  
POWER6®  
POWER7®  
POWER8®

PowerVM®  
Redbooks®  
Redbooks (logo) ®  
Storwize®  
System z®  
z/VM®

The following terms are trademarks of other companies:

SoftLayer, and SoftLayer device are trademarks or registered trademarks of SoftLayer, Inc., an IBM Company.

Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.



## Find and read thousands of IBM Redbooks publications

- ▶ Search, bookmark, save and organize favorites
- ▶ Get personalized notifications of new content
- ▶ Link to the latest Redbooks blogs and videos

Get the latest version of the Redbooks Mobile App



Download  
Now

Android



## Promote your business in an IBM Redbooks publication

Place a Sponsorship Promotion in an IBM® Redbooks® publication, featuring your business or solution with a link to your web site.

Qualified IBM Business Partners may place a full page promotion in the most popular Redbooks publications. Imagine the power of being seen by users who download millions of Redbooks publications each year!



[ibm.com/Redbooks](http://ibm.com/Redbooks)

About Redbooks → Business Partner Programs

THIS PAGE INTENTIONALLY LEFT BLANK

# Preface

This IBM® Redbooks® publication presents the IBM PowerKVM virtualization for scale-out Linux systems, including the new LC IBM Power Systems™.

PowerKVM is open source server virtualization that is based on the IBM POWER8® processor technology. It includes the Linux open source technology of KVM virtualization, and it complements the performance, scalability, and security qualities of Linux.

This book describes the concepts of PowerKVM and how you can deploy your virtual machines with the software stack included in the product. It helps you install and configure PowerKVM on your Power Systems server and provides guidance for managing the supported virtualization features by using the web interface and command-line interface (CLI).

This information is for professionals who want to acquire a better understanding of PowerKVM virtualization technology to optimize Linux workload consolidation and use the POWER8 processor features. The intended audience also includes people in these roles:

- ▶ Clients
- ▶ Sales and marketing professionals
- ▶ Technical support professionals
- ▶ IBM Business Partners
- ▶ Independent software vendors
- ▶ Open source community
- ▶ IBM OpenPower partners

It does not replace the latest marketing materials and configuration tools. It is intended as an additional source of information that, along with existing sources, can be used to increase your knowledge of IBM virtualization solutions.

Before you start reading, you must be familiar with the general concepts of kernel-based virtual machine (KVM), Linux, and IBM Power architecture.

## Authors

This book was produced by a team working at the International Technical Support Organization, Poughkeepsie Center.

**Murilo Opsfelder Araújo** is a Software Engineer working in the Linux Technology Center at IBM Brazil. He holds a Bachelor's degree in Computer Science from Anhanguera Educational Institute, Brazil. He is a Certified Linux Professional with experience in software development for Linux appliances and servers. He is also a Linux kernel hobbyist.

**Breno Leitao** is a Software Engineer at IBM in Brazil. He has been using Linux since 1997 and working in the Linux Technology Center at IBM since 2007. He holds a degree in Computer Science from the University of São Paulo. His areas of expertise include Linux, virtualization, networking, and performance.

**Stephen Lutz** is a Certified Senior Technical Sales Professional for Power Systems working for IBM Germany. He holds a degree in Commercial Information Technology from the University of Applied Science Karlsruhe, Germany. He is POWER8 champion and has 16 years experience in IBM AIX®, Linux, virtualization, and Power Systems and its predecessors, providing pre-sales technical support to clients, IBM Business Partners, and IBM sales representatives all over Germany. Stephen is also an expert in IBM PowerVC and IBM Cloud Manager with OpenStack with a focus on Power Systems.

**José Ricardo Ziviani** is a Software Engineer at IBM Brazil since 2010. During that time, he worked on the first versions of PowerKVM installer tool and is currently working on the Kimchi/Ginger project. His areas of interest are virtualization, software engineering, and electronics.

The project that produced this publication was managed by: **Scott Vetter, PMP**

Thanks to the following people for their contributions to this project:

Leonardo Augusto Guimaraes Garcia, Ricardo Marin Matinata, Gustavo Yokoyama Ribeiro, Fabiano Almeida Rosas, Lucas Tadeu Teixeira  
IBM Brazil

Gregory Kurz  
IBM France

Shivaprasad G Bhat, Bharata Bhasker Rao  
IBM India

Brian King, Frank P. Novak, Kersten J. Richter, George Romano  
IBM US

Thanks to the authors of the previous edition of this book.

- Authors of the first edition, *IBM PowerKVM: Configuration and Use*, published in October 2014, were:

Thadeu Lima de Souza Cascardo, Rafael Folco, Breno Leitao, Dinar Valeev

## Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

[ibm.com/redbooks/residencies.html](http://ibm.com/redbooks/residencies.html)

## Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

[ibm.com/redbooks](http://ibm.com/redbooks)

- Send your comments in an email to:

[redbooks@us.ibm.com](mailto:redbooks@us.ibm.com)

- Mail your comments to:

IBM Corporation, International Technical Support Organization  
Dept. HYTD Mail Station P099  
2455 South Road  
Poughkeepsie, NY 12601-5400

## Stay connected to IBM Redbooks

- Find us on Facebook:

<http://www.facebook.com/IBMRedbooks>

- Follow us on Twitter:

<https://twitter.com/ibmredbooks>

- Look for us on LinkedIn:

<http://www.linkedin.com/groups?home=&gid=2130806>

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>

- Stay current on recent Redbooks publications with RSS Feeds:

<http://www.redbooks.ibm.com/rss.html>



# Summary of changes

This section describes the technical changes made in this edition of the book and in previous editions. This edition might also include minor corrections and editorial changes that are not identified.

Summary of Changes  
for SG24-8231-01  
for IBM PowerKVM: Configuration and Use  
as created or updated on May 31, 2016.

## March 2016, Second Edition

This revision includes the following new and changed information.

### New information

- ▶ CPU Hotplug, see 5.4, “CPU Hotplug” on page 145
- ▶ Memory Hotplug, see 5.6, “Memory Hotplug” on page 157
- ▶ N\_Port ID Virtualization (NPIV), see 6.5, “N\_Port ID Virtualization (NPIV)” on page 176
- ▶ Usage of Multipathing, see 6.6, “Using multipath disks” on page 182
- ▶ Console configuration for Power LC Systems, see 2.4.2, “Console configuration for Power LC systems” on page 59
- ▶ PowerKVM Development Kit, see Chapter 8, “PowerKVM Development Kit” on page 227
- ▶ Security, see 7.4, “Security” on page 201
- ▶ Host management using Ginger, see 3.8, “Ginger” on page 96
- ▶ Docker, see 1.4, “Docker” on page 22

### Changed information

- ▶ Changes in the installer of PowerKVM
  - Manual installation
  - Automated installation
  - Host migration
  - Configuration tool
  - Console configuration for LC Power SystemsSee Chapter 2, “Host installation and configuration” on page 31
- ▶ Guest management using Kimchi, see Chapter 3, “Managing hosts and guests from a Web interface” on page 61
- ▶ Managing guests from the command-line interface, see Chapter 4, “Managing guests from the command-line interface” on page 105
- ▶ Update and enhancement of the following OpenStack related sections:
  - PowerVC
  - IBM Cloud Manager

- OpenStack controller services

See 7.5, “Cloud management” on page 204

- Generally, the content of the book has been updated to new attributes of the latest PowerKVM Version 3.1.





# Introduction

This chapter covers the concepts of open virtualization on IBM Power Systems. It introduces the IBM PowerKVM Version 3.1 virtualization stack and covers the following topics:

- ▶ Quick introduction to virtualization
- ▶ Introduction and basic concepts of PowerKVM
- ▶ IBM Power Systems
- ▶ IBM PowerKVM 3.1 software stack
- ▶ Docker and container concepts
- ▶ A comparison of IBM PowerKVM versions and IBM PowerVM®
- ▶ Terminology used throughout this book

## 1.1 IBM Power Systems

IBM Power Systems is a family of servers built for big data solutions by using advanced IBM POWER® processors. This family of servers includes scale-out servers and enterprise class servers from small to very large configurations. Power Systems are known for having high availability and extreme performance, among many other advantages that are covered later in this section.

Only a subset of these servers is covered in this book. This subset is referred as *IBM scale-out systems*, which include servers that run Linux-only operating systems that are based on the IBM POWER8 processor.

### 1.1.1 POWER8 processors

POWER8 is the most recent family of processors for Power Systems. Each POWER8 chip is a high-end processor that can have up to 12 cores, two memory controllers per processor, and a PCI generation 3 controller, as shown in Figure 1-1.

The processor also has a 96 MB of L3 shared cache plus 512 KB L2 cache per core.

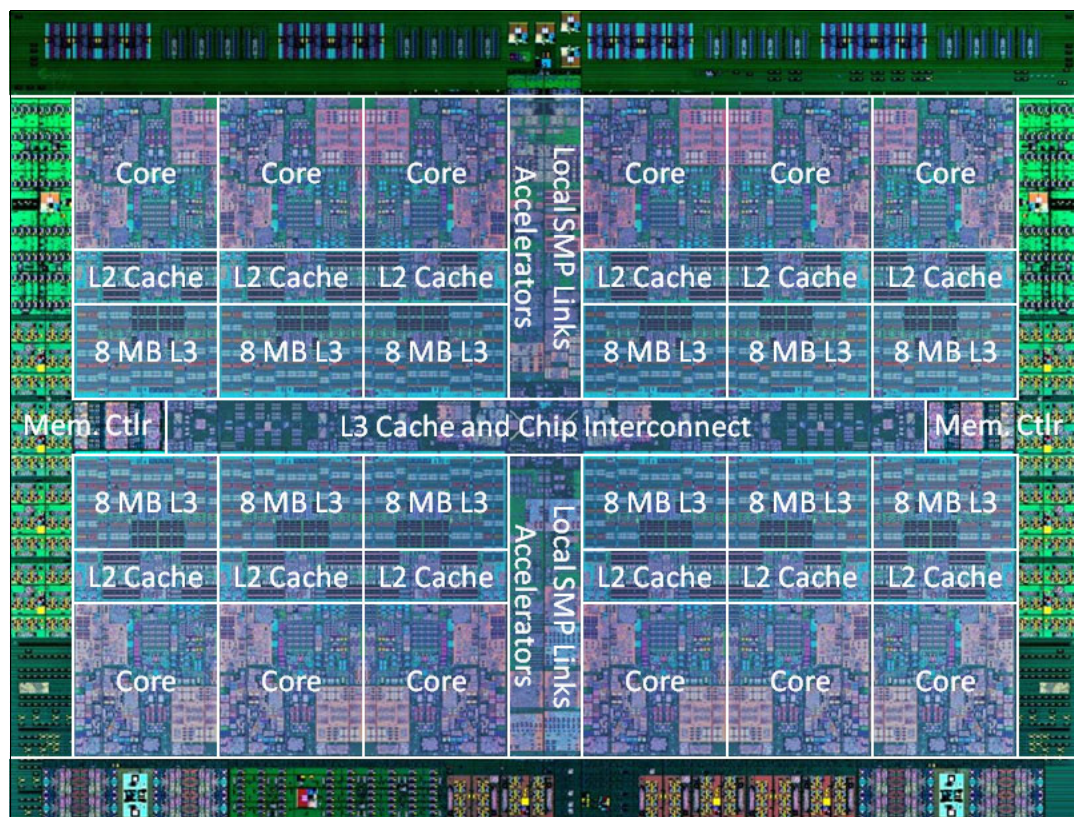


Figure 1-1 12-core IBM POWER8 architecture

The following features can augment performance of the IBM POWER8 processor:

- Support for DDR3 and DDR4 memory through memory buffer chips that offload the memory support from the IBM POWER8 memory controller

- ▶ L4 cache within the memory buffer chip that reduces the memory latency for local access to memory behind the buffer chip (the operation of the L4 cache is apparent to applications running on the POWER8 processor, and up to 128 MB of L4 cache can be available for each POWER8 processor)
- ▶ Hardware transactional memory
- ▶ On-chip accelerators, including on-chip encryption, compression, and random number generation accelerators
- ▶ Coherent Accelerator Processor Interface (CAPI), which allows accelerators plugged into a PCIe slot to access the processor bus by using a low-latency, high-speed protocol interface
- ▶ Adaptive power management
- ▶ Micro-Threading, which allows up to four concurrent VMs to be dispatched simultaneously on a single core

## 1.1.2 IBM scale-out servers

*IBM scale-out systems* are a family of servers built for scale-out workloads, including Linux applications that support a complete stack of open software, ranging from the hypervisor to cloud management.

The scale-out servers provide many benefits for cloud workloads, including security, simplified management, and virtualization capabilities. They are developed using open source methods.

At the time of publication, these are the base system models that are part of this family of servers:

- ▶ IBM Power System S812L (8247-21L)
- ▶ IBM Power System S822L (8247-22L)
- ▶ IBM Power System S824L (8247-42L)
- ▶ IBM Power System S812LC (8348-21C)
- ▶ IBM Power System S822LC (8335-GTA)
- ▶ IBM Power System S822LC (8335-GCA)

### IBM Power System S812L (8247-21L)

The S812L server, shown in Figure 1-2, is a powerful single-socket entry server. This server contains one POWER8 Dual-Chip Module (DCM) that offers 3.42 GHz (#EPLP4) or 3.02 GHz (#ELPD). This machine supports up to 12 IBM POWER8 cores, which provides up to 96 CPU threads when using SMT 8 mode.



Figure 1-2 Front view of the Power S812L



For more information about S812L, see *IBM Power Systems S812L and S822L Technical Overview and Introduction*, REDP-5098:

<http://www.redbooks.ibm.com/abstracts/redp5098.html?Open>

### IBM Power System S822L (8247-22L)

The S822L server, depicted in Figure 1-3, is a powerful two-socket server. This server supports up to two IBM POWER8 processors, so that offers 3.42 GHz (#ELP4) or 3.02 GHz (#ELPD) performance, reaching up to 24 cores and 192 threads when configured with SMT 8.



Figure 1-3 Front view of the Power S822L

Figure 1-4 shows the rear view of both models.



Figure 1-4 S821L and S822L rear view

### IBM Power System S824L

The IBM Power System S824L (8247-42L) server is a powerful four units server that supports up to 2 TB of memory. This server supports up to two IBM POWER8 processors, offering 4.15 GHz (#ELPH) or 3.52 GHz (#ELPJ) performance, reaching up to 24 cores and 192 threads when configured with SMT 8. This server also supports GPU adapters.

For more information about the IBM Power System S824L check *IBM Power System S824L Technical Overview and Introduction*, REDP-5139:

<http://www.redbooks.ibm.com/abstracts/redp5139.html?Open>



Figure 1-5 S824L front view

You can see the front and rear S824L view in Figure 1-5 and Figure 1-6.

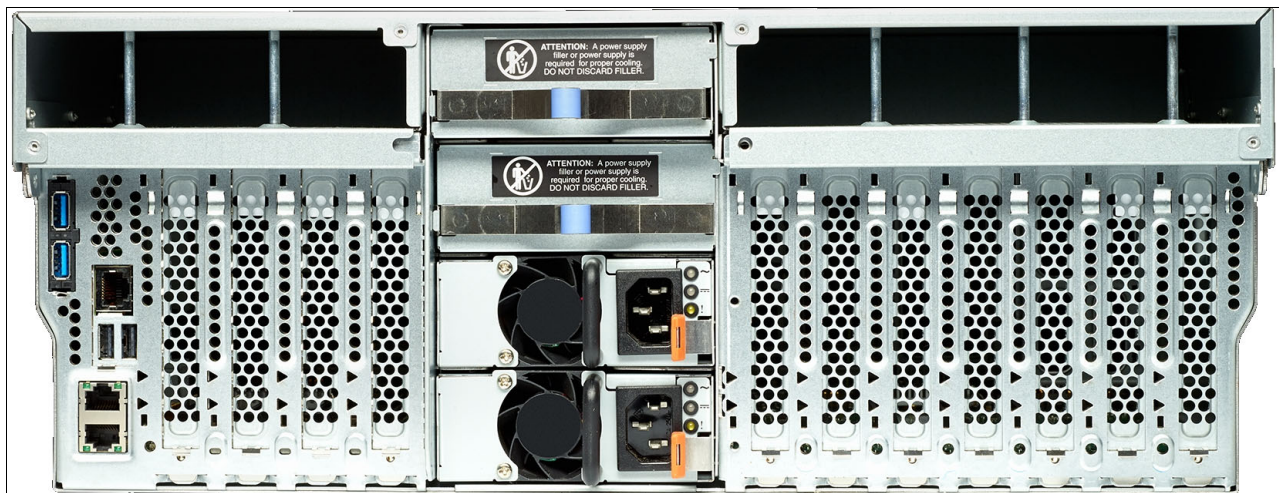


Figure 1-6 S824L rear view

The officially supported GPU for this specific machine is NVIDIA Tesla K40 based on the Kepler architecture. You can see this card at Figure 1-7 on page 6.

### **NVIDIA CUDA**

NVIDIA CUDA is a parallel computing platform and programming model that enables a dramatic increase in computing performance by offloading compute operations to the GPU card.

You can find details about the CUDA stack on POWER at Cuda Zone:

<https://developer.nvidia.com/cuda-tools-ecosystem>



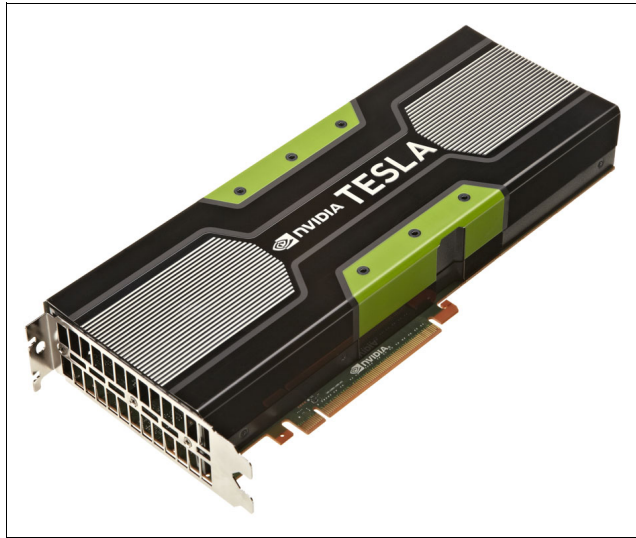


Figure 1-7 NVIDIA Tesla adapter

## IBM Power System S822LC

The IBM Power System S822LC servers are co-designed by the OpenPower foundation members focusing on industry standards and incorporate Open POWER Foundation members innovation. This machine is designed to run Linux.

The S822LC model is a two-unit server supporting up to two POWER8 processors, containing a total of either 16 or 20 2.92 GHz cores, and up to 1024 GB of memory. If SMT is enabled, this machine provides up to 160 hardware threads. Figure 1-8 shows the front view of the S822LC server.



Figure 1-8 Front view of the Power S822LC

### **8335-GCA**

The Power S822LC (8335-GCA) server supports two POWER8 processor sockets offering either eight 3.32 GHz cores (#EP00) or ten 2.92 GHz cores (#EP01) per socket.

### **8335-GTA**

The Power S822LC (8335-GTA) server supports two POWER8 processor sockets offering either eight 3.32 GHz cores (#EP00) or ten 2.92 GHz cores (#EP01) per socket. The 8335-GTA model also includes two NVIDIA K80 GPUs (#EC49).

For more information about S822LC servers, see *IBM Power System S822LC Technical Overview and Introduction*, REDP-5283:

<http://www.redbooks.ibm.com/abstracts/redp5283.html?Open>

## IBM Power System S8812LC12LC (8348-21C)

The IBM Power System S812LC is a one-processor socket server containing either eight 3.32 GHz or ten 2.92 GHz POWER8 cores and up to 1024 GB of memory.

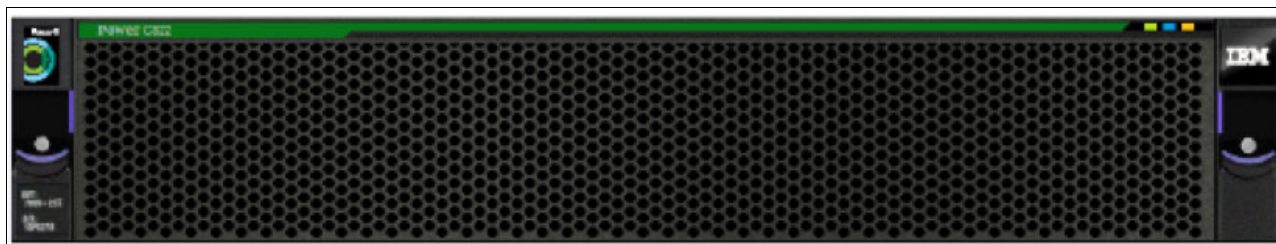


Figure 1-9 Front view of the Power S812LC

For more information about the S812LC server, see *IBM Power System S812LC Technical Overview and Introduction*, REDP-5284:

<http://www.redbooks.ibm.com/abstracts/redp5284.html?Open>

## Supported firmware

Table 1-1 shows the minimum firmware version required to support IBM PowerKVM V3.1 in each machine model.

Table 1-1 Recommended firmware version for PowerKVM

Machine Model	Firmware version
8247-21L	FW840
8247-22L	FW840
8247-42L	FW840
8348-21C	OP810
8335-GTA	OP810
8335-GCA	OP810

## 1.1.3 Power virtualization

IBM Power Systems servers have traditionally been virtualized with PowerVM, and this continues to be an option on model 8247 scale-out servers.

With the introduction of the Linux-only scale-out systems with POWER8 technology, a new virtualization mechanism is supported on Power Systems. This mechanism is known as a *kernel-based virtual machine* (KVM), and the port for Power Systems is called *PowerKVM*. You are not able to run both PowerVM and PowerKVM hypervisors at the same machine, at the same time.

KVM is known as the *de facto* open source virtualization mechanism. It is currently used by many software companies.

IBM PowerKVM is a product that leverages the Power resilience and performance with the openness of KVM, which provides several advantages:

- ▶ Higher workload consolidation with processors overcommitment and memory sharing
- ▶ Dynamic addition and removal of virtual devices
- ▶ Micro-Threading scheduling granularity
- ▶ Integration with IBM PowerVC, the IBM Cloud Manager with OpenStack, and native OpenStack
- ▶ Simplified management using open source software
- ▶ Avoids vendor lock-in
- ▶ Uses POWER8 hardware features, such as SMT8 and Micro-Threading
- ▶ NPIV support (*technology preview*)
- ▶ Docker support (*technology preview*)
- ▶ NVIDIA pass-through (*technology preview*)

For more information about IBM PowerKVM capabilities, check section 1.3.2, “KVM” on page 14.

**Note:** The IBM PowerVM and IBM PowerKVM hypervisors cannot be active at the same time on the same server.

## 1.1.4 Simultaneous multithreading

*Simultaneous multithreading* (SMT) is the ability of a single physical core to simultaneously dispatch instructions from more than one hardware thread context. Because there are eight threads per physical core, additional instructions can be executed in parallel. This hardware feature improves the overall efficiency of processor use.

In a POWER8 processor configured with SMT 8, up to 96 threads are available per socket, and each of them is represented as a processor in the Linux operating system.

## 1.1.5 Memory architecture

Power Systems based on the POWER8 processor uses a nonuniform memory access (NUMA) memory architecture. NUMA is a memory architecture on symmetric multiprocessing (SMP) systems. With this architecture, memory in different nodes has different access times, depending on the processor that is using it. The memory access can be local or remote, depending on whether the memory to be accessed is in the same core or in a different memory controller.

On the PowerKVM supported machines, there are up to three different memory nodes, and each of them has different performance when accessing other memory nodes.

Figure 1-10 on page 9 shows a two-socket server with a processor accessing local memory and another accessing remote memory.

PowerKVM is aware of the NUMA topology on the virtual machines, and tuning memory access might help the system performance.



To see the machine NUMA topology, use this command:

```
# virsh nodeinfo
```

For more information about memory tuning, see 5.3.4, “Configuring NUMA” on page 141.

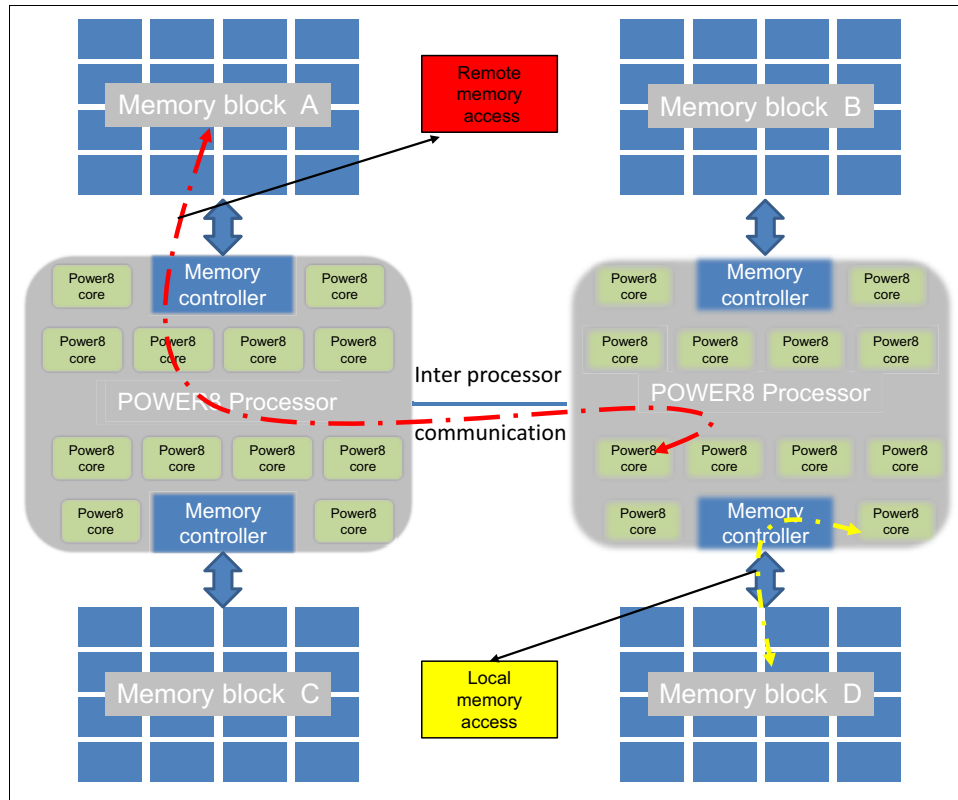


Figure 1-10 Local and remote memory access in a NUMA architecture

## 1.1.6 Micro-Threading

Micro-Threading is a POWER8 feature that enables a POWER8 core to be split into as many as two or four subcores. This gives the PowerKVM the capacity to support more than one virtual machine per CPU. Using Micro-Threading has many advantages when the virtual machine does not have enough workload to use a whole core. Up to four guests can run in parallel in the core.

Figure 1-11 shows the threads and the subcores on a POWER8 core when Micro-Threading is enabled and configured to support four subcores per core.

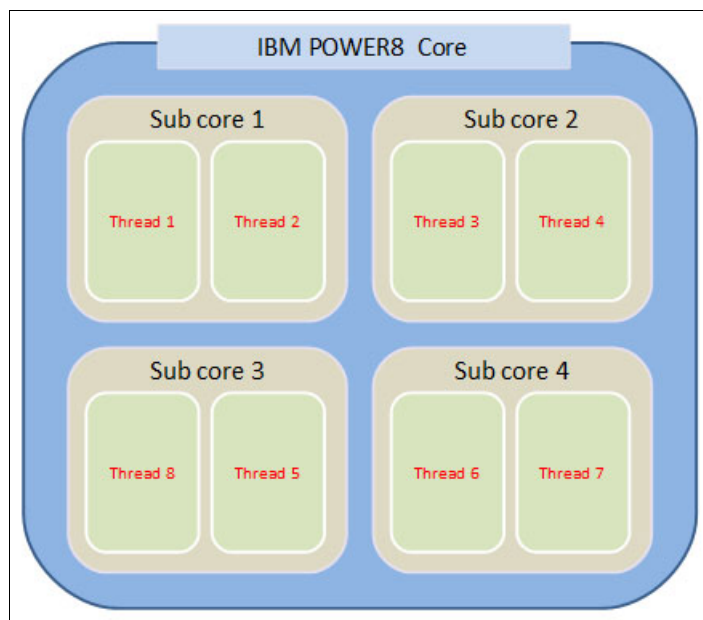


Figure 1-11 IBM POWER8 core with Micro-Threading enabled

### 1.1.7 RAS features

Power Systems are known for reliability, due to the servers' reliability, availability, and serviceability (RAS) features. These are some of the features that are available on IBM POWER8 servers:

- ▶ Redundant bits in the cache area
- ▶ Innovative ECC memory algorithm
- ▶ Redundant and hot-swap cooling
- ▶ Redundant and hot-swap power supplies
- ▶ Self-diagnosis and self-correction of errors during run time
- ▶ Automatic reconfiguration to mitigate potential problems from suspected hardware
- ▶ Self-heal or automatically substitute functioning components for failing components

**Note:** Some servers might have additional unique RAS features, and servers such as the LC servers, use industry standard RAS features. It is best to check the features on the server that you intend to deploy.

For more information about the Power Systems RAS features, see *IBM Power Systems S812L and S822L Technical Overview and Introduction*, REDP-5098.

<http://www.redbooks.ibm.com/abstracts/redp5098.html?Open>

## 1.2 Virtualization

For practical purposes, this publication focuses on server virtualization, especially ways to run an operating system inside a virtual machine and how this virtual machine acts. There are many advantages when the operating system runs in a virtual machine rather than in a real machine, as later sections of this book explain.

With virtualization, there are two main pieces of software, the hypervisor and the guest:

- Hypervisor

The *hypervisor* is the operating system and firmware that runs directly on the hardware and provides support. One traditional example of a hypervisor for Power Systems is the PowerVM server virtualization software.

- Guest

*Guest* is the usual name for the virtual machine. As Figure 1-12 illustrates, a guest always runs inside a hypervisor. In the PowerVM and IBM System z® world, a guest is called a *logical partition* (LPAR).

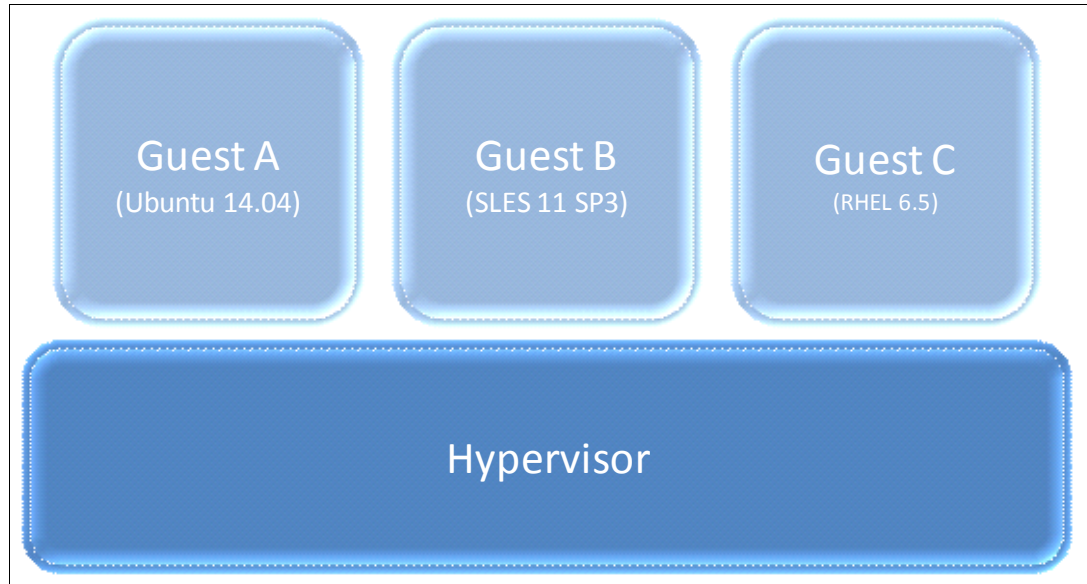


Figure 1-12 Guests and hypervisor

**Notes:** The terms *virtual machine* and *guest* are used interchangeably in this book.

## 1.2.1 PowerKVM versions

Table 1-2 shows the available versions of the PowerKVM stack.

Table 1-2 PowerKVM versions

Product name	Program ID	Sockets
IBM PowerKVM 2.1	5765-KVM	up to 2
IBM PowerKVM 2.1, 1-year maintenance	5771-KVM	up to 2
IBM PowerKVM 2.1, 3-year maintenance	5773-KVM	up to 2
IBM PowerKVM 3.1	5765-KV3	up to 2
IBM PowerKVM 3.1, 1-year maintenance	5771-KVM	up to 2
IBM PowerKVM 3.1, 3-year maintenance	5773-KVM	up to 2

## 1.2.2 PowerKVM Version 3.1 considerations

There are several new considerations in the PowerKVM V3.1 release:

- ▶ PowerKVM Version 3.1 is based on the POWER ABI version 2. The previous version was based on Power Abstract Binary Interface (ABI) version 1. ABI version 1 saves data in big endian mode, while ABI version 2 uses the little endian order.
- ▶ PowerKVM does not support IBM AIX or IBM i operating systems.
- ▶ PowerKVM cannot be managed by the Hardware Management Console (HMC).
- ▶ The SPICE graphic model is not supported.
- ▶ PowerKVM supports a subset of the I/O adapters (as PowerKVM is developed, the adapter support continually changes)
- ▶ Only one operating system is allowed to run on the host system. PowerKVM does not provide multiboot support.

### PowerKVM guest limits

Table 1-3 lists the guest limits, or the maximum amount of resources to assign to PowerKVM virtual machines.

*Table 1-3 Maximum amount of resources per virtual machine*

Resources	Recommended maximum
Guest memory	512 GB
Guest vCPUs	160
Virtual network devices	8
Number of virtual machines per core	20
Number of para-virtualized devices	32 PCI device slots per virtual machine and 8 PCI functions per device slot

## 1.2.3 Where to download PowerKVM

You can order a Power machine with PowerKVM preinstalled on the system or you can order a physical media and install it yourself. There is also the possibility to download installation image from an IBM website.

The following product images are available to download:

- |                                |   |
|--------------------------------|---|
| <b>Full installation image</b> | A bootable image containing the full set of packages needed to install PowerKVM.  |
| <b>Update image</b>            | A non-bootable image containing only the updated packages. It cannot be used to install PowerKVM. It can only be used to update a current installation. |
| <b>Debuginfo image</b>         | A non-bootable image containing all debuginfo packages. This image can be useful for debugging purposes.  |
| <b>Source image</b>            | A non-bootable image containing all source packages of the product.   |

The full installation images are only available through the Entitled Systems Support (ESS) website:

<http://www.ibm.com/servers/eserver/ess/index.wss?lnk=msdD0-enss-usen>

**Note:** Make sure that you registered your system and have your customer number handy.

The update, debuginfo, and source images are available from the IBM Fix Central website:

<https://www.ibm.com/support/fixcentral>

After you download the full installation image from ESS page, you can use it to install PowerKVM from a NetBoot server or burn it on a DVD to perform a local installation. Both methods are covered in Chapter 2, “Host installation and configuration” on page 31.

## 1.3 Software stack

This section covers all of the major software in a common IBM PowerKVM deployment.

### 1.3.1 QEMU

QEMU is an open source software that hosts the virtual machines on a KVM hypervisor. It is the software that manages and monitors the virtual machines and performs the following basic operations:

- ▶ Create virtual image disks
- ▶ Change the state of a virtual machine:
  - Start virtual machine
  - Stop a virtual machine
  - Suspend a virtual machine
  - Resume a virtual machine
  - Take and restore snapshots
  - Delete a virtual machine
- ▶ Handle the I/O between guests and the hypervisor
- ▶ Migrate virtual machines

In a simplified view, you can consider the QEMU as the *user* space tool for handling virtualization and KVM the *kernel* space module.

QEMU can also work as an emulator, but that situation is not covered in this book.

#### QEMU monitor

QEMU provides a virtual machine monitor that helps control the virtual machine and performs most of the operations required by a virtual machine. The monitor can inspect the virtual machine low-level operations, such as details about the CPU registers, I/O device states, ejects a virtual CD, and many other things.

You can use the command shown in Figure 1-13 to see the block devices that are attached to a QEMU image.

Figure 1-13 QEMU monitor window example

```
(qemu) info block
scsi0-hd0: image.qcow2 (qcow2)

scsi0-cd2: utopic-server-ppc64el.iso (raw, read-only)
Removable device: locked, tray closed
(qemu) █
```

To run a QEMU monitor by using the **virsh** command, use the following parameters:

```
# virsh qemu-monitor-command --hmp <domain> <monitor command>
```

## 1.3.2 KVM

A *kernel-based virtual machine* (KVM) is a part of open source virtualization infrastructure that turns the Linux kernel into an enterprise-class hypervisor.

QEMU is another part of this infrastructure, and KVM is usually referred as the QEMU and KVM stack of software. Throughout this publication, *KVM* is used as the whole infrastructure on the Linux operating system to turn it into a usable hypervisor.

The whole stack used to enable this infrastructure is described in 1.3, “Software stack” on page 13.

KVM requires hardware virtualization extensions, as described in section “Hardware-assisted virtualization” on page 131.

### KVM performance

Because KVM is a very thin layer over the firmware, it can deliver an enterprise-grade performance to the virtual machines and can consolidate a huge amount of work on a single server. One of the important advantages of virtualization is the possibility of using *resource overcommitment*.

#### **Resource overcommitment**

*Overcommitment* is a mechanism to expose more CPU, I/O, and memory to the guest machine than exists on the real server, thereby increasing the resource use and improving the server consolidation.

#### **SPEC performance**

KVM is designed to deliver the best performance on virtualization. There are virtualization-specific benchmarks. Possibly the most important one at the moment is called *SPECvirt*, which is part of the Standard Performance Evaluation Corporation (SPEC) group.

SPECvirt is a benchmark that addresses performance evaluation of data center servers that are used in virtualized server consolidation. It measures performance of all of the important components in a virtualized environment, from the hypervisor to the application running in the guest operating system.

For more information about the benchmark as the KVM results, check the SPEC web page:

[http://www.spec.org/virt\\_sc2013](http://www.spec.org/virt_sc2013)

## 1.3.3 Open Power Abstraction Layer

IBM Open Power Abstraction Layer (OPAL) is a small layer of firmware that is available on POWER8 machines. It provides support for the PowerKVM software stack, as well as for bare metal mode.

The OPAL firmware contains two major internal components:

- Skiboot

Skiboot is the runtime and the main booting service

- Skirroot

Skirroot is the boot loader application that is called when the machine is booted. Skirroot contains two other parts, the kernel and the *initramfs*. Initramfs contain other components, such as *petitboot*.

OPAL is part of the firmware that interacts with the hardware and exposes it to the PowerKVM hypervisor. It comes in several parts, and this is an example about how it works on an OpenPower server:

1. The baseboard management controller (BMC) is responsible to power on the system.
2. The BMC starts to boot each chip individually using the Self Boot Engine (SBE) part.
3. When the processors are started, the BMC calls Flexible Service Interface (FSI), that is the primary service interface in the POWER8 processor. There is a direct connect between the BMC and FSI called Low Pin Count (LPC).
4. After that, the hostboot firmware IPLs the system, using a secondary power-on sequence called Digital Power System Sweep (DPSS).
5. At that time, the hostboot firmware loads the OPAL and moves all the CPUs to starting point.

Opal development is done on public GitHub community. You can clone the code using the following repository:

<https://github.com/open-power>

### 1.3.4 Guest operating system

The following operating systems are supported as guests in the PowerKVM environment:

- Red Hat Enterprise Linux Version 6.5 or later  
<http://www.redhat.com/products/enterprise-linux>
- SUSE Linux Enterprise SLES 11 SP3 or later  
<https://www.suse.com/products/server>
- Ubuntu 14.04 or later  
<http://www.ubuntu.com>

There are also other community Linux distributions that technically run as guests on PowerKVM, such as:

- Debian  
<http://www.debian.org>
- Fedora  
<https://getfedora.org>
- OpenSuse  
<http://www.opensuse.org>
- CentOS  
<https://www.centos.org/>

### 1.3.5 Libvirt software

Libvirt software is the open source infrastructure to provide the low-level virtualization capabilities in most hypervisors that are available, including KVM, Xen, VMware, IBM PowerVM. The purpose of libvirt is to provide a more friendly environment for the users.

Libvirt provides different ways of access, from a command line called **virsh** to a low-level API for many programming languages.

The main component of the libvirt software is the *libvirtd* daemon. This is the component that interacts directly with QEMU or the KVM software.

This book covers only the command-line interface. See chapter Chapter 4, “Managing guests from the command-line interface” on page 105. There are many command-line tools to handle the virtual machines, such as **virsh**, **guestfish**, **virt-df**, **virt-clone**, **virt-xfstool**, and **virt-image**.

For more Libvirt development information, see Chapter 8, “PowerKVM Development Kit” on page 227 or check the online documentation:

<http://libvirt.org>

### 1.3.6 Virsh interface

Virsh is the command-line interface used to handle the virtual machines. It works by connecting to the libvirt API that connects to the hypervisor software (QEMU in the PowerKVM scenario) as shown in Figure 1-14. Virsh is the main command when managing the hypervisor using the command line.

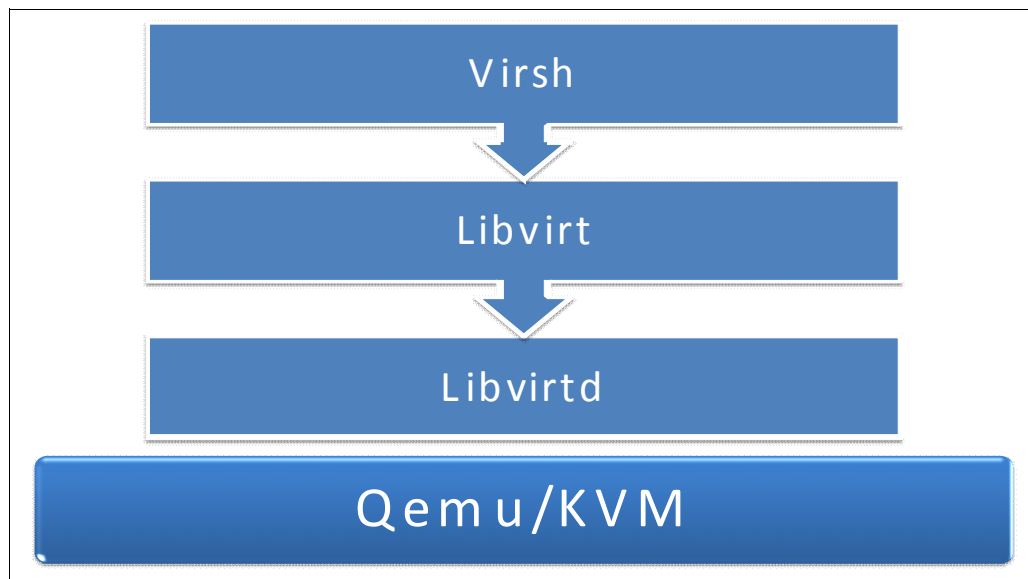


Figure 1-14 Virsh and libvirt architecture

For more information about **virsh**, see Chapter 4, “Managing guests from the command-line interface” on page 105.

#### Virtual machine definition

A virtual machine can be created directly by using **virsh** commands. All the virtual machines managed by virsh are represented by an XML file. It means that all the virtual machine settings, as the amount of CPU and memory, disks and I/O are defined in an XML file.



For more information, see Chapter 4, “Managing guests from the command-line interface” on page 105.

### 1.3.7 Intelligent Platform Management Interface

Intelligent Platform Management Interface (IPMI) is an out-of-band communication interface and a software implementation that controls the hardware operation. It is a layer below the operating system, so the system is still manageable even if the machine does not have an operating system installed. It is also used to get the console to install an operating system in the machine.

On *S8nnL* models, the IPMI server is hosted in the service processor controller. On *S8nnLC* models, it is hosted in the BMC. This means that the commands directed to the server should use the service processor IP address, not the hypervisor IP address.

These are some of the IPMI tools that work with PowerKVM:

- ▶ OpenIPMI
- ▶ FreeIPMI
- ▶ IPMITool

Table 1-4 shows the most-used IPMI commands.

*Table 1-4 Frequently used IPMI commands*

Command	Description
<b>sol</b>	Serial over LAN console
<b>power</b>	Control the machine power state
<b>sensor</b>	Show machine sensors, as memory and CPU faults
<b>fru</b>	Information about machine Field Replaceable Units (FRUs) parts
<b>user</b>	Configure user accounts on IPMI server

**Note:** The IPMI protocol is based on UDP protocol, which means that it might lose datagram. Prefer to use it through lossless network routes.

### 1.3.8 Petitboot

Petitboot is an open source platform independent boot loader based on Linux. It is used in the PowerKVM hypervisor stack, and is used to boot the hypervisor operating system.

Petitboot includes graphical and command-line interfaces, and can be used as a client or server boot loader. For this document, only the basic use is covered.

```
Petitboot (70fb2ec) 8205-E6C 06A22ER
*IBM_PowerKVM, with Linux 3.10.23-1500.pkvm2_1.5.ppc64
IBM_PowerKVM, with Linux 0-rescue-306415a668f240b0b0c4b5974bb35891
POWERKVM_LIVECD

System information
System configuration
Exit to shell
```

Figure 1-15 Petitboot window

For more information about this software, check the Petitboot web page:

<https://www.kernel.org/pub/linux/kernel/people/geoff/petitboot/petitboot.html>

You are also able to find Petitboot source code publicly available at OpenPower GitHub:

<https://github.com/open-power/petitboot>

### 1.3.9 Kimchi

Kimchi is a local web management tool meant to manage a few guests virtualized with PowerKVM. Kimchi has been integrated into PowerKVM and allows initial host configuration, as well as managing virtual machines using the web browser through an HTML5 interface.

The main goal of Kimchi is to provide a friendly user interface for PowerKVM, allowing them to operate the server by using a browser most of the time. These are some of the other Kimchi features:

- ▶ Firmware update
- ▶ Backup of the configuration
- ▶ Host monitoring
- ▶ Virtual machine templates
- ▶ VM guest console
- ▶ VM guest VNC
- ▶ Boot and install from a data stream

To connect to the Kimchi page, the browser should be pointed to the hypervisor using the HTTPS port 8001. Figure 1-16 shows the Kimchi home panel.

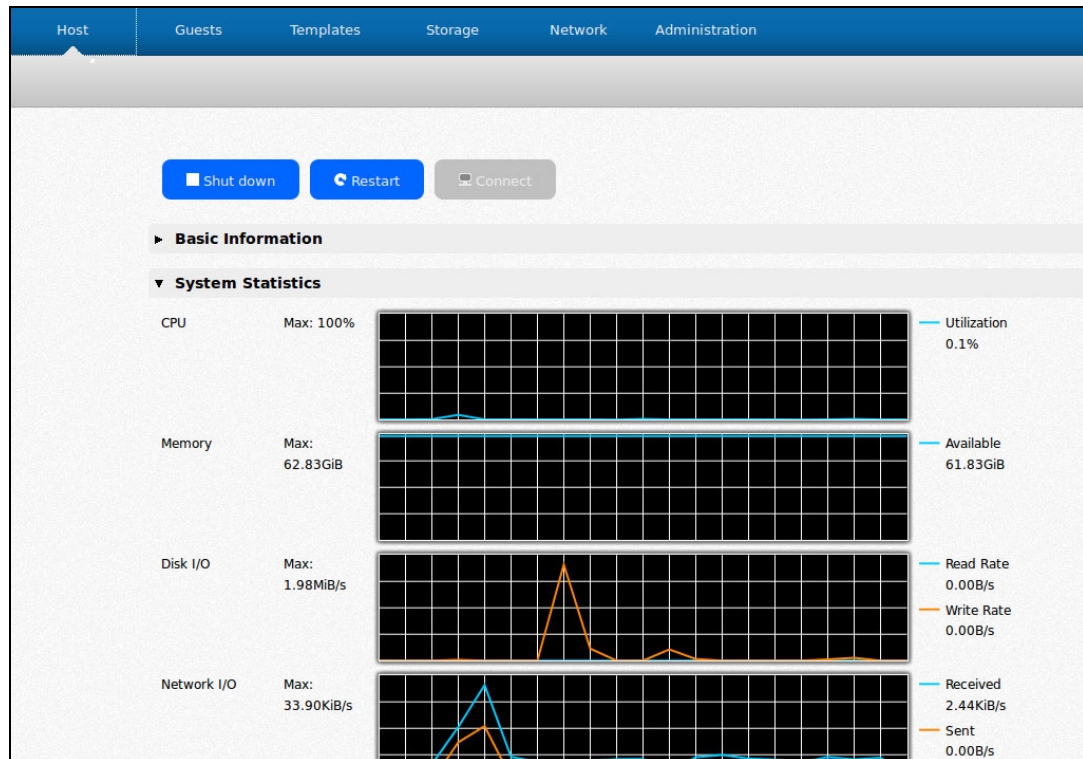


Figure 1-16 Kimchi home panel

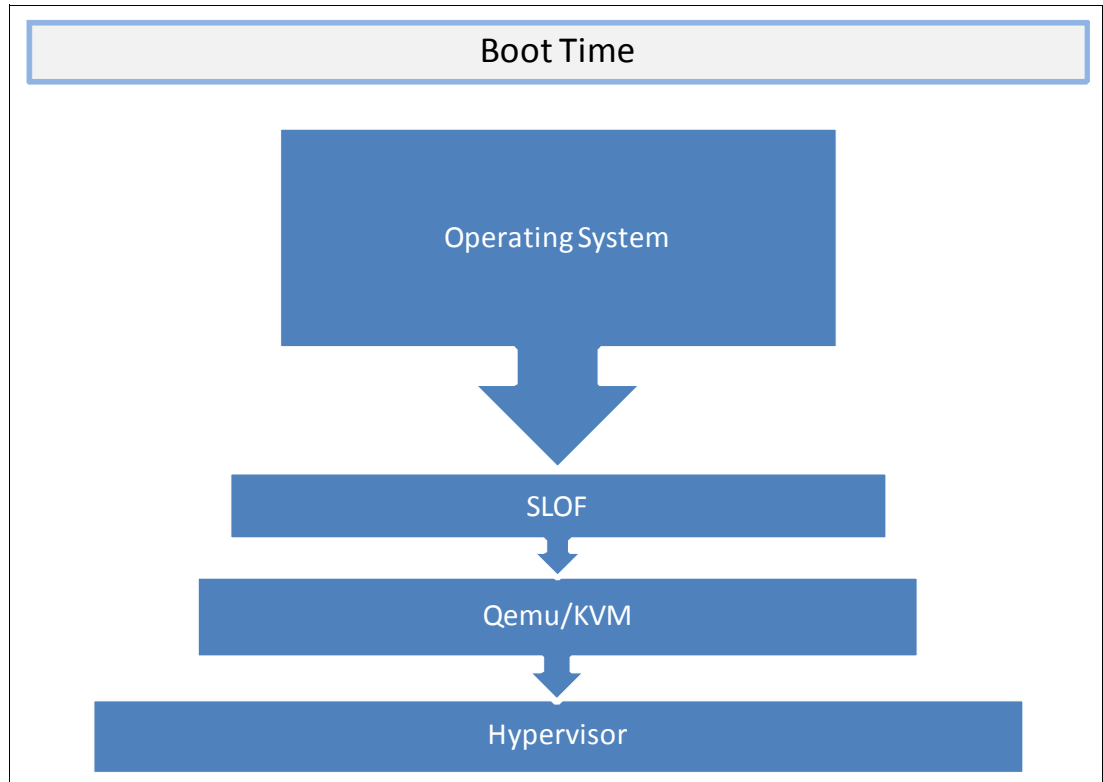
For more information about the Kimchi project, see Chapter 3, “Managing hosts and guests from a Web interface” on page 61 or the project web page:

<http://github.com/kimchi-project/kimchi>

### 1.3.10 Slimline Open Firmware

Slimline Open Firmware (SLOF) is an open source firmware used on PowerKVM to boot the guest OS.

SLOF is also a machine independent firmware based on the IEEE-1275 standard, also known as the *Open Firmware Standard*. It executes during boot time and then it is not necessary any more in the system, so it is removed from the memory. An abstraction of the SLOF architecture is shown in Figure 1-17.



*Figure 1-17 SLOF during VM boot*

For more information, see the SLOF web page:

<http://www.openfirmware.info/SLOF>

### 1.3.11 Virtio drivers

Virtio is a virtualization standard that enables high-performance communication between guests and the hypervisor. This is based on the guest being virtual machine-aware and, as a result, cooperating with the hypervisor.

The general architecture is shown in Figure 1-18.

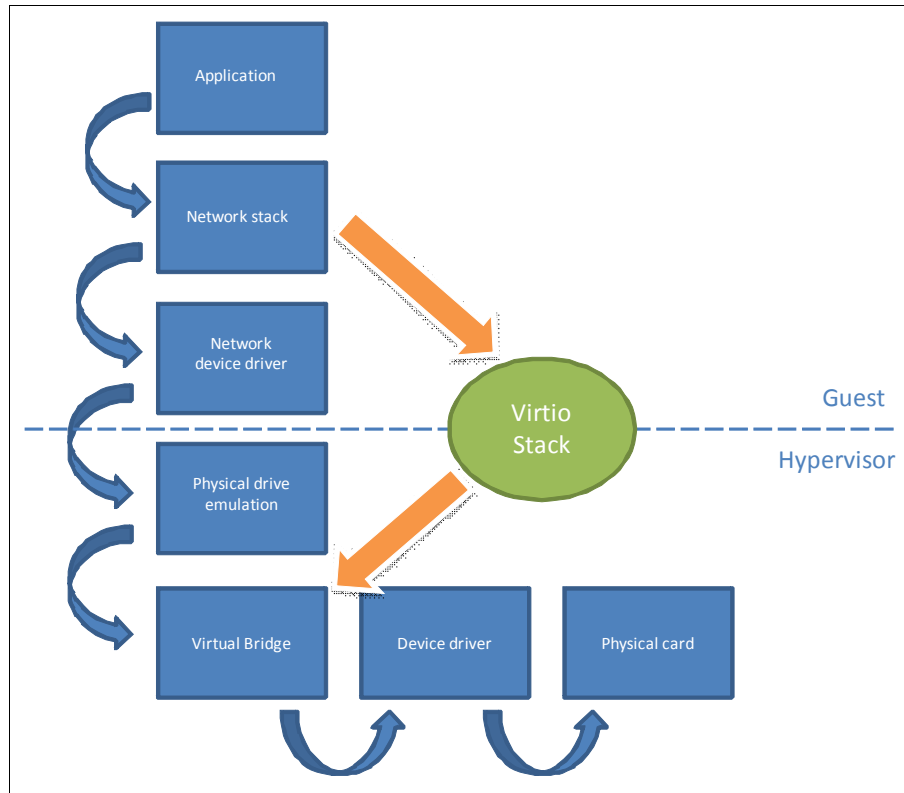


Figure 1-18 Virtio architecture for network stack

Many Virtio drivers are supported by QEMU. The main ones are provided in Table 1-5.

Table 1-5 Virtio drivers

Device driver	Description
virtio-blk	Virtual device for block devices
virtio-net	Virtual device for network driver
virtio-pci	Low-level virtual driver to allow PCI communication
virtio-scsi	Virtual storage interface that supports advanced SCSI hardware
virtio-balloon	Virtual driver that allows dynamic memory allocation
virtio-console	Virtual device that allows console handling
virtio-serial	Virtual device driver for multiple serial ports
virtio-rng	Virtual device driver that exposes hardware number generator to the guest

For the guest point of view, the drivers need to be loaded in the operating system.

**Note:** ibmveth and ibmvscsi are also paravirtualized drivers.

### 1.3.12 RAS stack

The RAS tools is a set of applications to manage Power Systems that are part of the PowerKVM stack. This set of tools work on low-level configurations. The primary goal is to change the POWER processor configuration, such as enabling and disabling SMT, enabling Micro-Threading, discovering how many cores are configured in the machine, determining the CPU operating frequency and others. The following tools are a subset of the packages in `powerpc-utils`:

- ▶ `ppc64_utils`
- ▶ `drmgr`
- ▶ `lsslot`
- ▶ `update_flash`
- ▶ `bootlist`
- ▶ `opal_errd`
- ▶ `lsvpd`
- ▶ `lsmcode`

## 1.4 Docker

Docker is a software application that creates an operating system abstraction layer able to run independent and isolated containers on top of it. This technology creates an environment similar to virtualization and it is usually called a *lightweight hypervisor*. Docker is based on three underline major Linux features called *namespaces*, *change root (chroot)*, and *control groups (cgroups)*.

Docker also provides an official image repository with plenty of ready-to-use and freely distributed container images. Download these images takes no more than one single Docker command.

### Linux namespace

*Linux namespace* is an isolation concept where each namespace has the illusion that it has control of the whole system, and it is not able to see the content of any other namespace. There are several types of namespaces. Basically, namespaces exist for all major subsystems as process, users, IPC, network, mount tables, and so on.

Figure 1-19 shows an example of two namespaces. There are two namespaces: **A** and **B**. Namespace **A** has a limited access to the process list and mount points, while namespace **B** contains another set of processes and mount points. They do not interfere with each other. They are not even able to query that there are other namespaces.

Different processes using the same PID can exist handling different processes on different namespaces. This is what allows different namespaces to run different initialization processes as a traditional *System V init.d* boot process as depicted in namespace **A**, and a *Systemd* boot process, as in namespace **B**.

Global Process List View		
PID	Process	Namespace
1	Init	A
1	Systemd	B
2	Apache	A
3	Sshd	B
2	Nginx	B

Global mount table View		
Mountpoint	device	Namespace
/	/dev/sda	A
/	192.168.1.1/root	B
/home	/dev/sdb	A
/tmpfs	tmpfs	B
/home	192.168.1.1/home	B

Namespace A	
PID	Process
1	Init
2	Apache
Mountpoint	device
/	/dev/sda
/home	/dev/sdb

Namespace B	
PID	Process
1	Systemd
3	Sshd
2	Nginx
Mountpoint	device
/	192.168.1.1/root
/tmpfs	Tmpfs
/home	192.168.1.1/home

Figure 1-19 Namespaces example

## Change root

*Change root* is a Linux feature that changes the file system root directory for a specific process, hence the process file system directories are restricted for that process, similarly to process *namespace* but aiming at file system restriction. This feature also gives the process the illusion to see the whole file system and access any directory, but in this case, that is not real, the access happens inside a jail, and the process is not allowed to access any file outside of that jail.

Figure 1-20 shows an example of a *change root* feature and the jail environment. On the bottom, you see the jail environment. It has a completely new file system hierarchy inside a normal directory. The *chrooted* process is only able to access files and directories inside that jail. The *chrooted* process will have the illusion that the root directory is the system directory, but it is, in fact, a simple subdirectory.

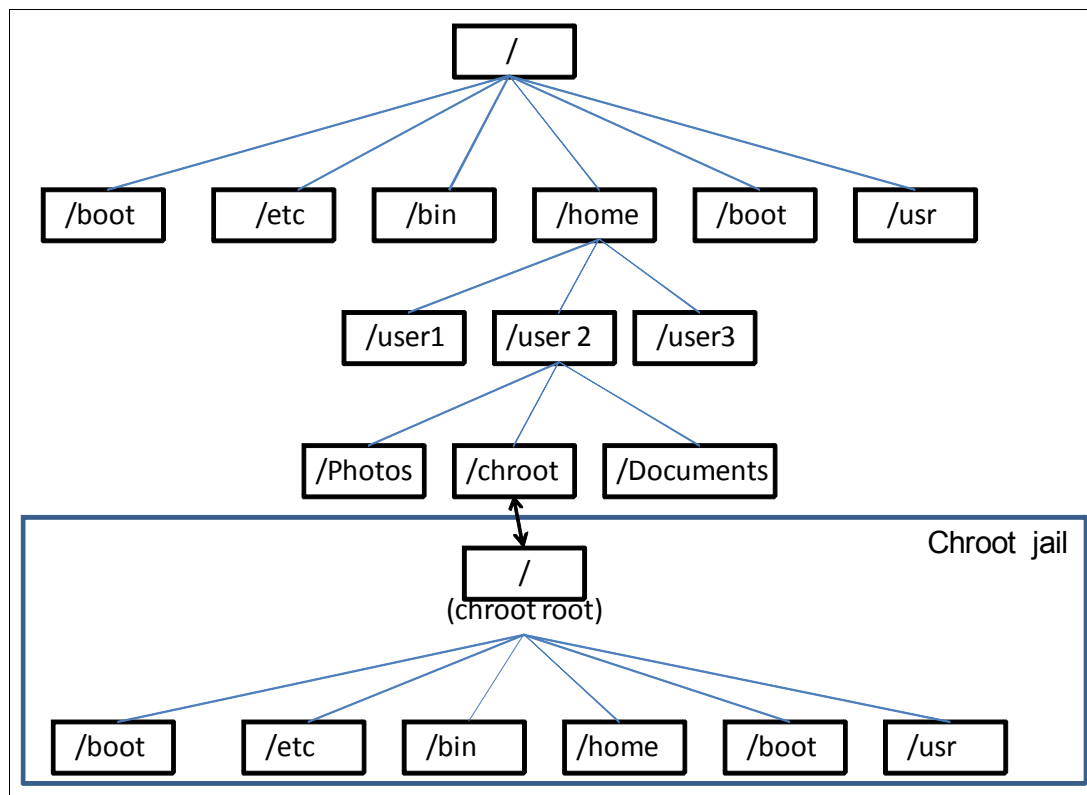


Figure 1-20 Change root feature

## Control groups

*Control groups* is a feature that accounts and limits resource utilization for a specific set of processes. This is the kernel feature that limits each container to not use more memory, CPU, and I/O than specified by the hypervisor administrator.

Other than just limiting the resources for a set of processes, control groups is also responsible for prioritizing requests for CPU utilization, I/O throughput, and memory bandwidth.

Another important feature provided by control groups is the ability to checkpoint and restart a userspace set of applications, allowing a process (or container) to be stopped in a certain machine, and restart in another one. This is the concept behind container migration.



## 1.4.1 Architecture

IBM PowerKVM Version 3.1 supports Docker as a technology preview. Docker on PowerKVM creates an interesting virtualization scenario where the user is able to create virtual machines and containers in the same hypervisor and they are able to coexist at the same time.

Figure 1-21 depicts a situation where two Docker containers and two virtual machines are running at the same time.

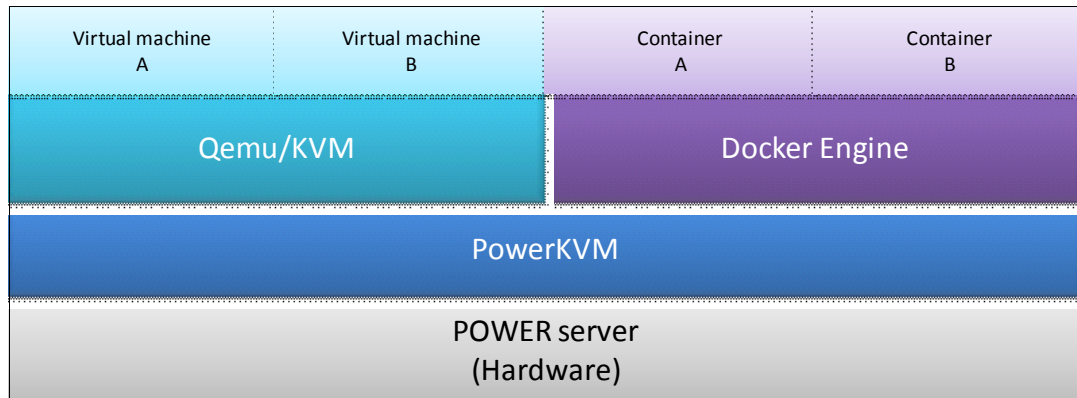


Figure 1-21 Containers and virtual images on a POWER server

Although a container and a virtual machine can coexist in the same environment, they are managed using different tool sets and there is not a simple way to see them together. The Libvirt community started a project to manage virtual machines and Docker containers in a libvirt extension named *libvirt-lxc*, but it is not production ready yet. When available, virtual machines are managed by kimchi or libvirt, while containers are managed by the Docker command line.

Other than that, a container must run using the hypervisor kernel, in this case a PowerKVM kernel. Hence, a container cannot have a different and unique kernel, as it happens with virtual machines. Container architecture allows the container to run in the userspace stack in a restricted environment on top of the PowerKVM kernel.

Qemu and KVM allow a full new operating system instantiation, allowing any kind of kernel, and thus any software stack, to run on top of an emulated hardware way, as shown in Figure 1-22.

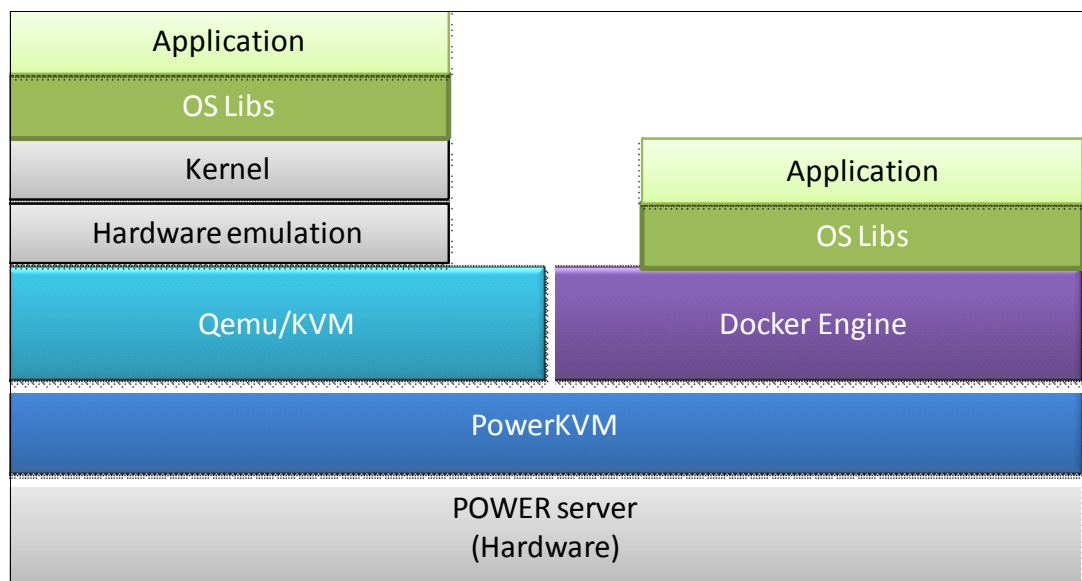


Figure 1-22 Container and guest OS stack

## Differences between container and virtual machines

There are differences between the technology used to create virtual machines and containers, as already listed in Table 1-6.

The container technology has less overhead and software flexibility when compared to a full virtualization.

Table 1-6 Differences between containers and virtual machines

Feature	Virtual machine	Container
Kernel version restriction	No	Yes
Ready-to-use image repository	No	Yes
Fine-grained resource limitation	Yes	No
Support Emulated I/O	Yes	No
Management stack	Libvirt, Kimchi, virsh	Docker
Multiples instances per image	No	Yes
File system type	qcow, raw	aufs
Virtualization level	System	Application

## 1.4.2 Docker hub

Docker hub is a container image service that provides ready-to-use container images to Docker. Each container image can contain a single application, or even a complex environment already configured by the image uploader. Any Docker user can upload and download an image to Docker hub.

Docker hub became the standard way to distribute container images in the Docker community. If you want to have any freely distributed software running on top of a Docker container, you probably will find it in Docker hub, and it would take you one command to download it and get it running on your system.

Docker hub can be accessed directly from the Docker application. To search for an image, you can use the **docker search** command. If you want to download the image, you can use **docker pull**. To upload an image, the command **docker push** is used.

For more information about these commands, see 7.6.4, “Uploading your image to Docker hub” on page 222.

### 1.4.3 Docker file

Docker file is a document that specifies how a Docker image is created. A Docker image is usually defined as a set of scripts and commands that runs on top of a single and base image. A Docker file basically transforms a base image, as a stripped Linux image, into a final workload provider.

Figure 1-23 shows the process of creating a Docker image from a Docker file. When the Docker file is converted into an image, you can start this image and in this case, you have a unique container. Multiple containers can point to the same image, but each container forks the original image and creates forked images.

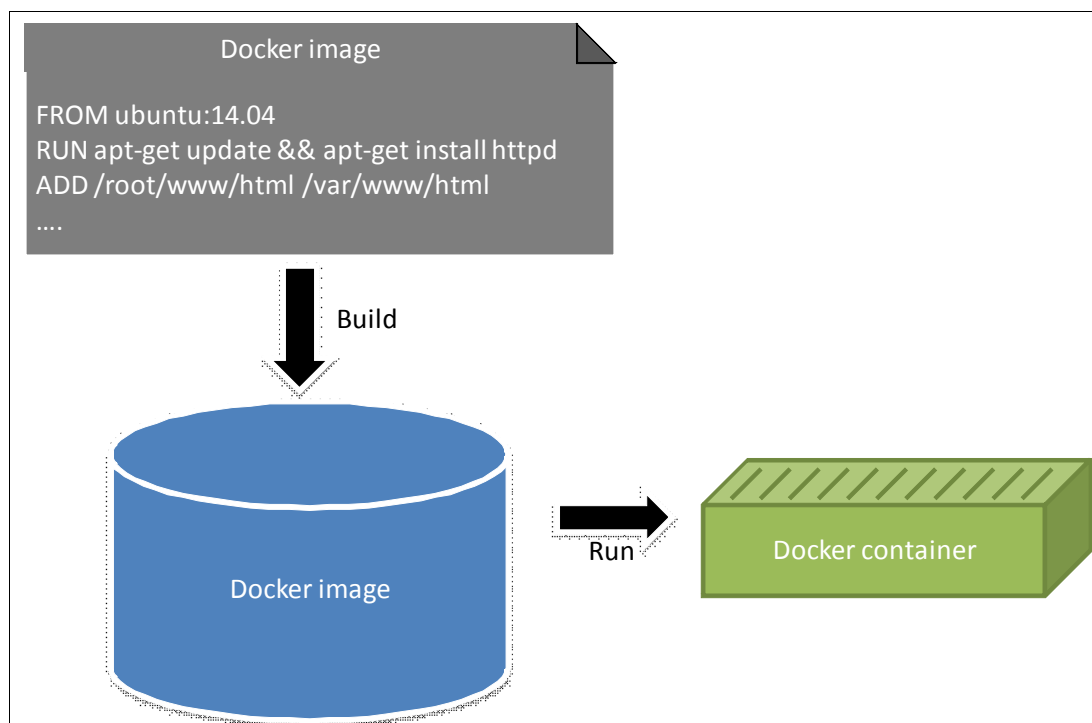


Figure 1-23 Build and run process

## 1.5 Comparisons of PowerVM and PowerKVM

This section covers the comparison between PowerVM and PowerKVM, and also this PowerKVM version and the previous one.

## 1.5.1 PowerVM and PowerKVM features

Table 1-7 compares IBM PowerVM and IBM PowerKVM features.

*Table 1-7 Comparison of IBM PowerKVM and IBM PowerVM features*

Feature	IBM PowerVM	IBM PowerKVM
Adding devices to the guest	Dynamic LPAR	Hot plug
Containers	Yes <sup>a</sup>	Yes, as a technology preview
Different editions	Yes (Standard, and Enterprise)	No
Dynamic logical partition	Yes	Yes
Guests running in Big and Little Endian	Yes	Yes
License	Proprietary	Open source
Live partition mobility	Yes	Yes
Memory compression	Yes (IBM Active Memory™ Expansion)	No (zswap can be installed manually)
Memory page sharing	Yes (Described as Active Memory Data Deduplication)	Yes (Described as Kernel SamePage Merging, or KSM)
IBM Micro-Partitioning®	Yes	Yes
NPIV	Yes	Yes, as a technology preview
PCI pass-through	Yes	Yes
Shared storage pools	Yes	Yes
Sparse disk storage	Yes (thin provisioning)	Yes (qcow2 image)
Supported machines	All non-LC IBM Power Systems	IBM scale-out systems only
Supported operating systems in the guest	IBM AIX IBM i Linux	Linux

a. PowerVM has a similar concept for AIX only named WPAR

## 1.5.2 PowerKVM Version 2.1 and Version 3.1

Table 1-8 shows a comparison between the two PowerKVM releases considering the major features.

*Table 1-8 PowerKVM releases comparison*

Feature	IBM PowerKVM V2.1	IBM PowerKVM V3.1
ABI version	1	2
Automated Dynamic Micro-threading	No	Yes
CPU hot plug	No	Yes

Feature	IBM PowerKVM V2.1	IBM PowerKVM V3.1
Development Kit	No	Yes
Docker on the host	No	Yes ( <i>technology preview</i> )
Endianness mode	Big Endian	Little Endian
GPU pass-through	No	Yes ( <i>technology preview</i> )
Kernel version	3.10	3.18
Libvirt version	1.2.5	1.2.16
Memory hot plug	No	Yes
NPIV	No	Yes ( <i>technology preview</i> )
OpenStack version	Icehouse	Liberty
Supported machines	S822L S812L	S821IL S822L S812LC S822LC S824L

## 1.6 Terminology

Table 1-9 lists the terms used for PowerKVM and the counterpart terms for KVM on x86.

Table 1-9 Terminology comparing KVM and PowerVM

IBM PowerKVM	KVM on x86	IBM PowerVM
Flexible service processor (FSP) Baseboard Management Controller (BMC)	Integrated management module (IMM)	Flexible service processor (FSP)
guest, virtual machine	guest, virtual machine	LPAR
hot plug	hot plug	dynamic LPAR
hypervisor, host	hypervisor, host	hypervisor
Image formats: qcow2, raw, nbd, and other image formats	acow2, raw, nbd, and other image formats	proprietary
IPMI	IPMI	HMC
kernel samepage merging (KSM)	kernel samepage merging	Active Memory Data Deduplication
Kimchi and virsh	Kimchi and virsh	HMC and Integrated virtualization manager (IVM)
KVM host user space (QEMU)	KVM host user space (QEMU)	Virtual I/O Server (VIOS)
Open Power Abstraction Layer (OPAL)	Unified Extensible Firmware Interface (UEFI) and BIOS	PowerVM hypervisor driver (pHyp)
Preboot Execution Environment (PXE)	Preboot eXecution Environment	BOOTP and TFTP, NIM
Slimline Open Firmware (SLOF)	SeaBIOS	Open Firmware, SMS
Virtio drivers, ibmvscsi, and ibmveth	Virtio drivers	ibmvscsi, ibmveth
Virtual Network Computing (VNC)	VNC	None
zswap	zswap	Active Memory Expansion (AME)



# Host installation and configuration

This chapter covers IBM PowerKVM installation and configuration.

After reading this chapter, you should be able to perform the following tasks:

- ▶ Install PowerKVM from a local DVD media
- ▶ Automate installations over a network
- ▶ Reinstall PowerKVM
- ▶ Migrate a host from an older release
- ▶ Configure an installed system

## 2.1 Host installation

This section describes how to install PowerKVM on the host. You also have the option to order a PowerKVM system preinstalled.

PowerKVM supports physical DVD media and NetBoot installation methods. The OPAL firmware uses Petitboot, a kexec-based bootloader capable of loading kernel and initrd from any Linux mountable file system.

PowerKVM is available in ISO format only. For DVD installation, burn the ISO image file into a DVD media. To set up a NetBoot installation, you must extract the NetBoot files from the ISO image on the boot server, as shown in Figure 2-18 on page 42.

**Note:** PowerKVM cannot be installed on a SAN disk, such as a Fibre Channel connected device. However, SAN disks can be used as storage backing for guest images and other file systems.

To start installing PowerKVM on your system, follow these steps:

1. Insert the DVD or configure NetBoot on another system to install from a remote server. See Figure 2-18 on page 42 for more details about how to configure NetBoot installation. The instructions that follow apply to both installation methods.
2. Connect to the system through the serial port or by using Serial Over LAN (SOL) by using an Intelligent Platform Management Interface (IPMI) network console. An IPMI console is provided by the *ipmitool* package from most of the Linux distributions and also for Windows, such as the IPMIUTIL available on:

<http://ipmiutil.sourceforge.net>

**Important:** For scale-out Power Systems, the IPMI network console must be enabled with a password on the ASM interface. See 2.4.1, “Console configuration for Scale-out Power Systems” on page 58 for more information.

For the Power LC Systems, the default password is *admin* to connect to the system.

- To activate *SOL*, use the following command:

```
$ ipmitool -I lanplus -H <FSP/BMC IP ADDRESS> -P <IPMI PASSWORD> sol  
activate
```

- To deactivate *SOL*, use the following command:

```
$ ipmitool -I lanplus -H <FSP/BMC IP ADDRESS> -P <IPMI PASSWORD> sol  
deactivates
```

3. Boot the system and wait for the Petitboot menu to display (Figure 2-1 on page 33). You can use the ASMI interface for scale-out Power Systems or the IPMI command line to power-on the system. To power the system on or off, run the following commands:

```
$ ipmitool -I lanplus -H <FSP/BMC IP ADDRESS> -P <IPMI PASSWORD> power on
```

```
$ ipmitool -I lanplus -H <FSP/BMC IP ADDRESS> -P <IPMI PASSWORD> power off
```

```
$ ipmitool -I lanplus -H <FSP/BMC IP ADDRESS> -P <IPMI PASSWORD> power cycle
```



```
Petitboot (dev.20141013)

POWERKVM_LIVECD

System information
System configuration
Language
Rescan devices
Retrieve config from URL
*Exit to shell

Enter=accept, e=edit, n=new, x=exit, l=language, h=help
Welcome to Petitboot
```

Figure 2-1 Petitboot menu

4. Wait for the PowerKVM installer to automatically start.

You can also choose the installation method from the Petitboot menu. If you are installing from DVD media, choose the **POWERKVM\_LIVECD** entry from the menu options. For NetBoot installations, choose the menu entry label that was previously configured in the NetBoot server or select *Retrieve config from URL* to point the installer to the necessary resources.

**Note:** Petitboot starts the installer automatically after a 10-second timeout. You can also change or disable the automatic boot on a persistent basis by changing the *Autoboot* option on the System configuration menu.

Petitboot uses DHCP as default for the network configuration. Use the **System configuration** option from the Petitboot menu to change the network settings as shown in Figure 2-2. You can also choose to skip this step and configure the network after the installation is complete. See “Configure the network” on page 56 for more details.

Petitboot System Configuration

Autoboot:
( ) Don't autoboot
(\*) Autoboot from any disk/network device
( ) Only autoboot from a specific disk/network device

Timeout:
10 seconds

Network:
( ) DHCP on all active interfaces
(\*) DHCP on a specific interface
( ) Static IP configuration

Device:
( ) eth0 [00:0e:1e:80:60:84, link down]
( ) eth1 [00:0e:1e:80:60:85, link down]
( ) eth2 [00:0e:1e:80:60:86, link down]
( ) eth3 [00:0e:1e:80:60:87, link down]
(\*) eth4 [40:f2:e9:5d:4f:e0, link up]
( ) eth5 [40:f2:e9:5d:4f:e1, link down]
( ) eth6 [40:f2:e9:5d:4f:e2, link down]

tab=next, shift+tab=previous, x=exit, h=help

Figure 2-2 Petitboot system configuration

- Choose the language. English, as shown in Figure 2-3, is used for this installation.

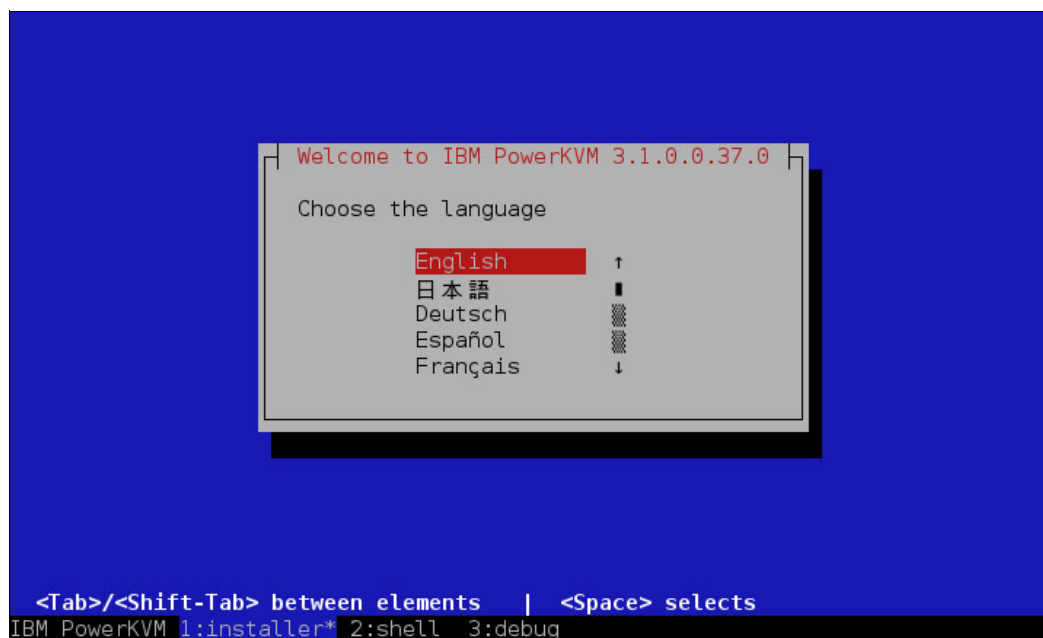


Figure 2-3 Language selection

**Tip:** The PowerKVM installer allows you to switch from panels during the installation process. For example, if you want to switch to the *shell* panel, use CTRL-B and then type 2.

6. After reading and accepting the license agreement, select **Install PowerKVM**, click **OK**, and then press Enter.



Figure 2-4 Welcome to IBM PowerKVM

**Note:** If the installer detects an existing PowerKVM system on one of the disks, it also offers to do an Install over existing IBM PowerKVM, which preserves the guest images. For more information about reinstalling PowerKVM, see 2.3, "Install over existing IBM PowerKVM and host migration" on page 52.

The option Install over existing IBM PowerKVM is for migrating from PowerKVM 2.1 to PowerKVM V3.1. For more information about migrating PowerKVM, also refer to section 2.3, "Install over existing IBM PowerKVM and host migration" on page 52.

7. Select the target device to install. A PowerKVM installation requires a minimum of 38 GB of disk space and additional space for the images. In the dialog box (Figure 2-5), you can select multiple disks for the installation. It is required that at least one disk is a local disk. Additional disks that can be out of the SAN can be used as storage for the images.

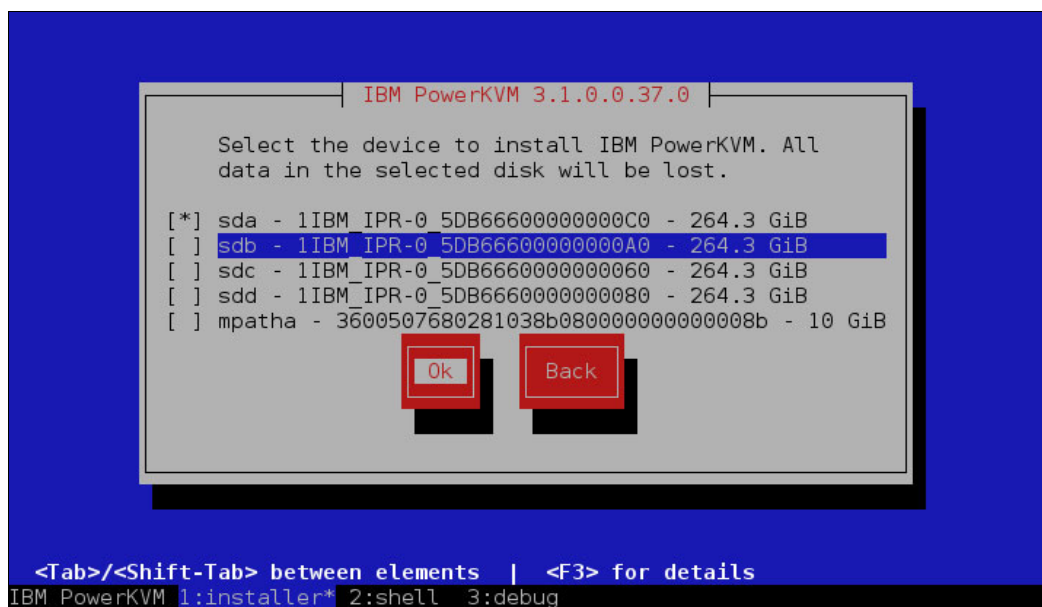


Figure 2-5 Select the target device to install PowerKVM

8. In the next step, as shown in Figure 2-6, the name of the volume group can be defined. The default is `ibmpkvm_vg_root`. Changing the name of the volume group can be useful for example if you want to install several instances of PowerKVM on different disks inside one system for demo purposes.

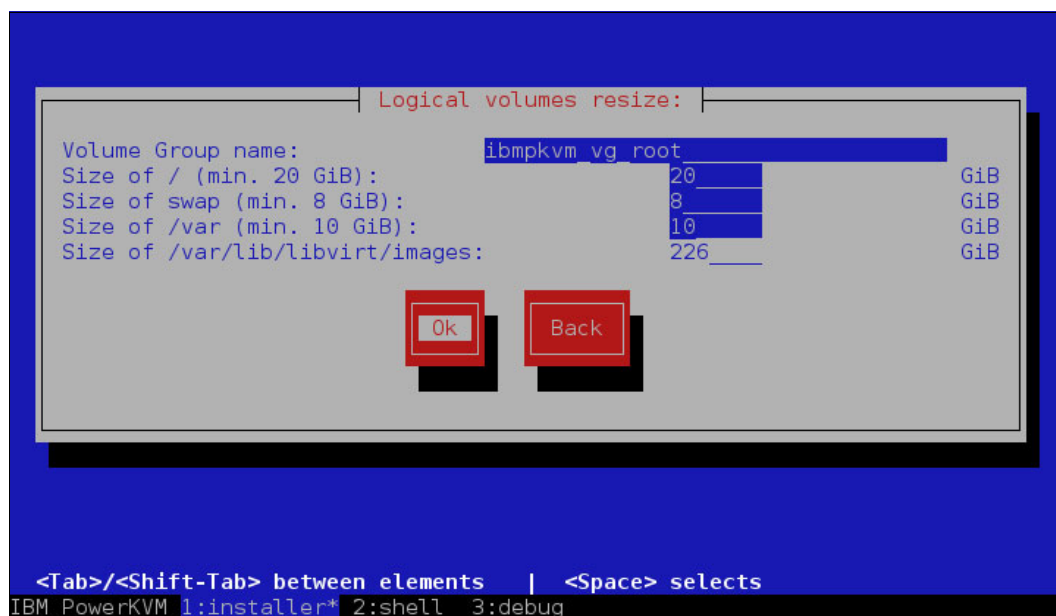


Figure 2-6 Change the name of the volume group and the size of the volumes if needed

9. Set a password for the root user. The password must have at least six characters. Choose a password that conforms with your site security rules (Figure 2-7).

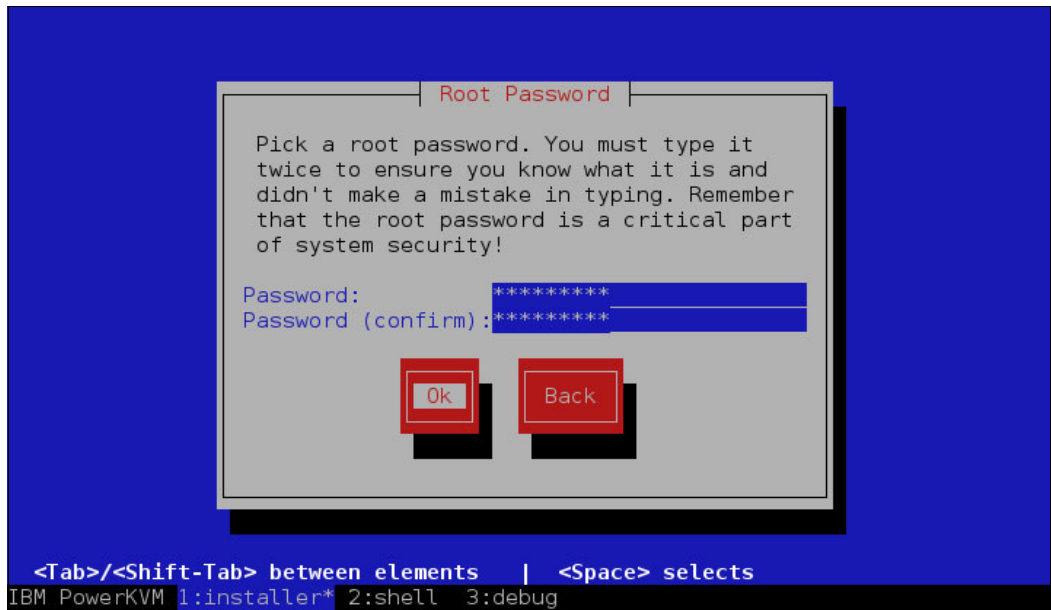


Figure 2-7 Root password

10. Set the time zone for the system. Select the indicated box if the system clock uses UTC (Figure 2-8).

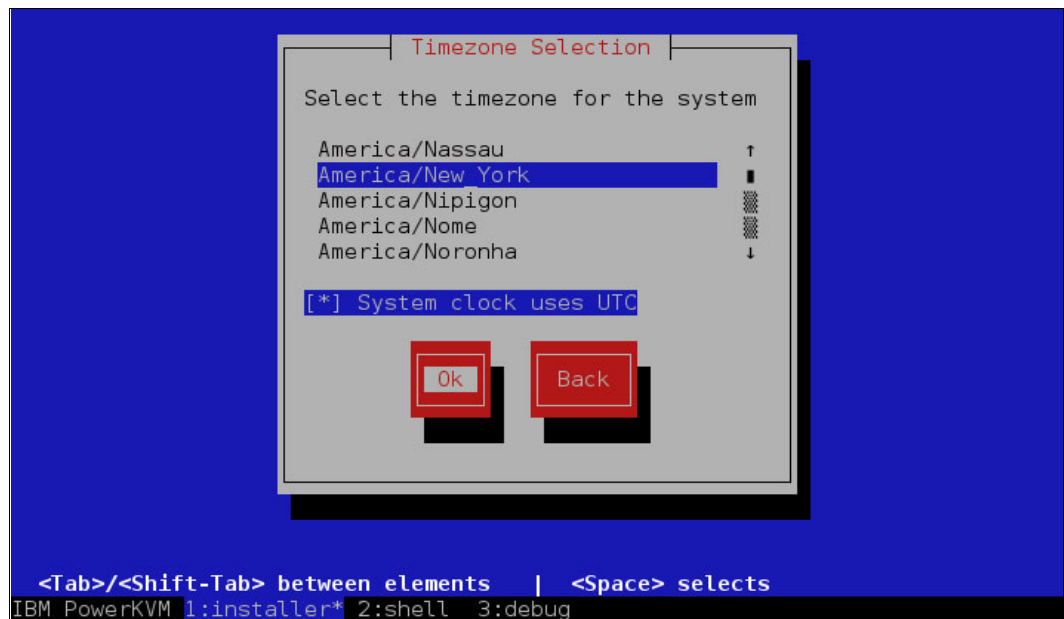


Figure 2-8 Time zone selection

11. Configure the NTP servers as shown in Figure 2-9. Use a time provider that is reachable within your network topology.

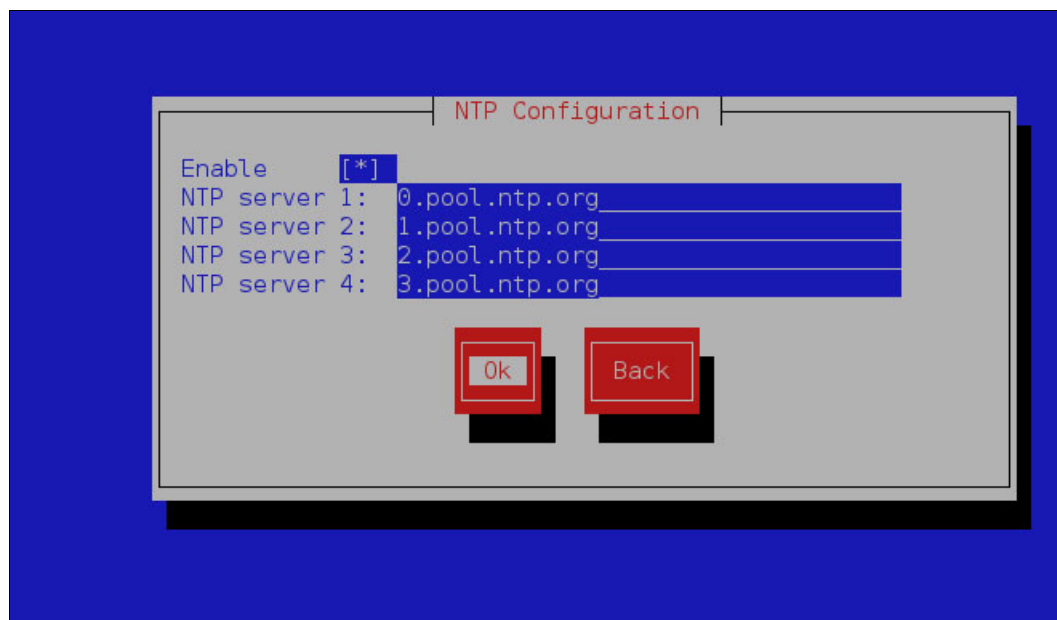


Figure 2-9 NTP configuration

**Important:** Setting up NTP servers is a preferred practice to keep the clock synchronized between host and guests.

12. Set the date and time. Figure 2-10 shows an example when this publication was written.

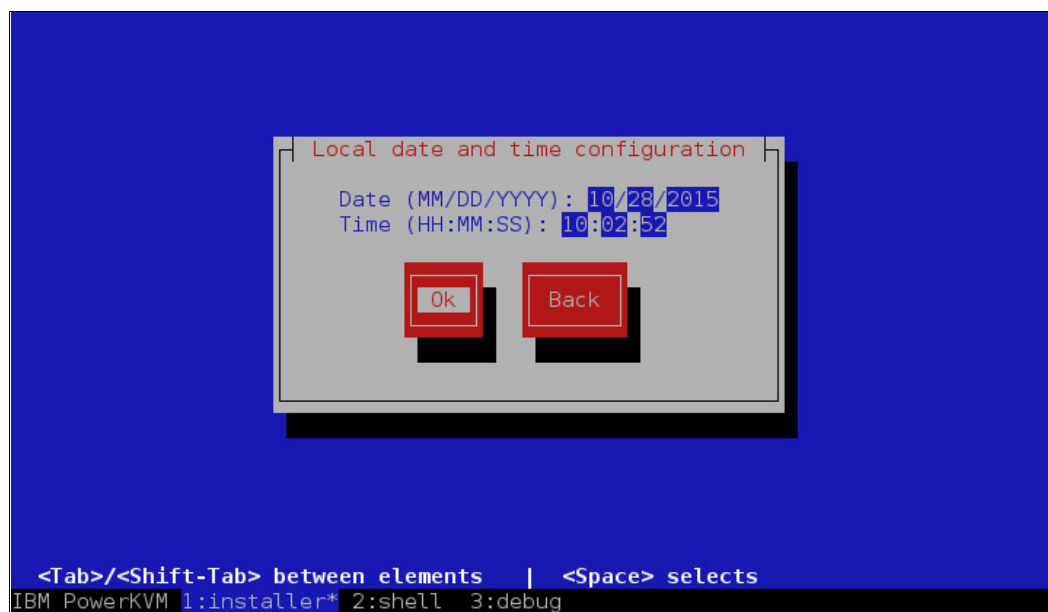


Figure 2-10 Date and time configuration

13. Select the network devices that require configuration. Figure 2-11 shows the list of network devices to be configured.

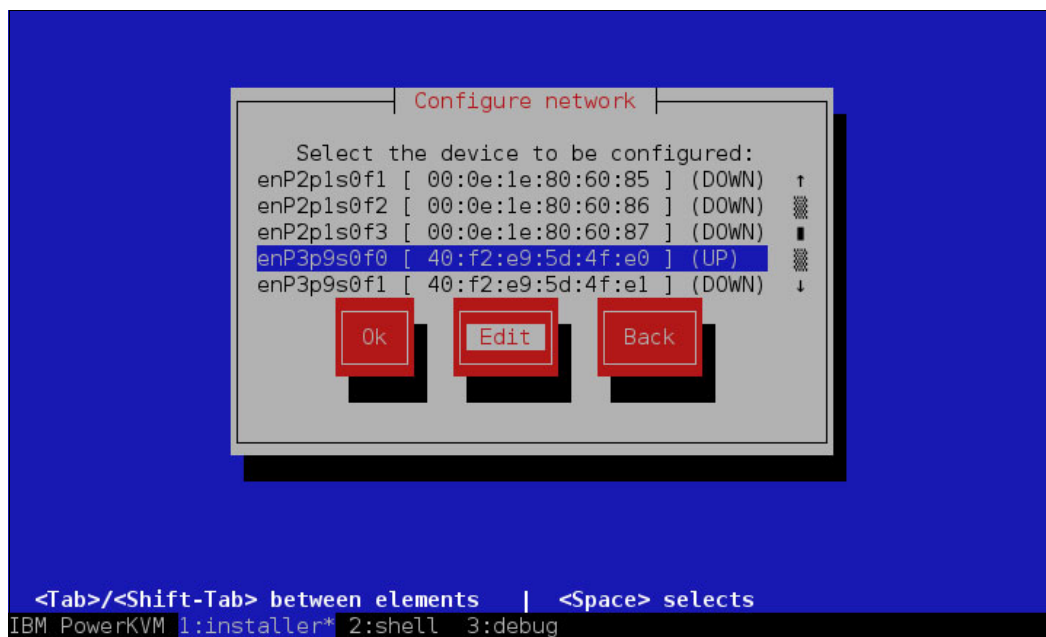


Figure 2-11 Configure network

14. You can edit network devices by selecting the **Edit** option, as shown in Figure 2-12.

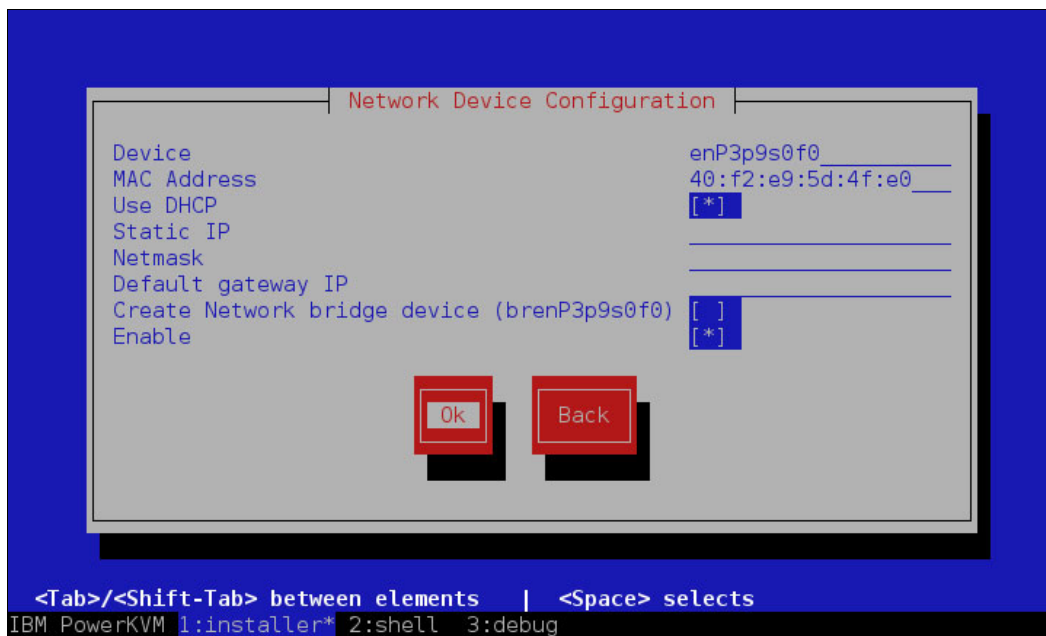


Figure 2-12 Network device configuration

15. Configure the DNS, as shown in Figure 2-13. Having a reliable name service provides a trouble-free installation.

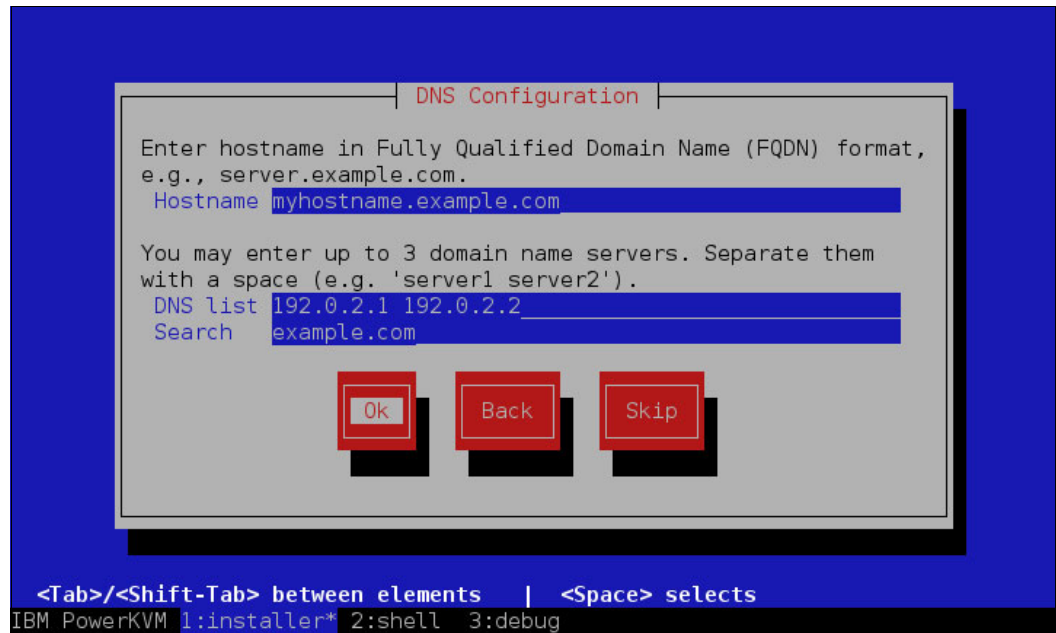


Figure 2-13 DNS configuration

16. Review the installation summary as shown in Figure 2-14.

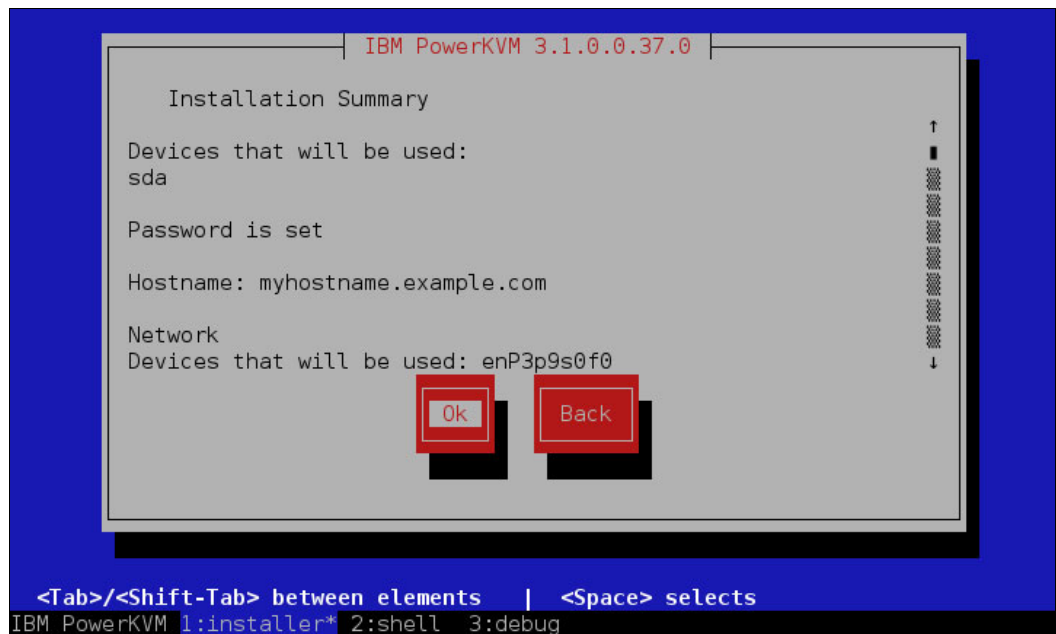


Figure 2-14 Installation summary



17. Confirm the device that you have chosen to install, as shown in Figure 2-15.

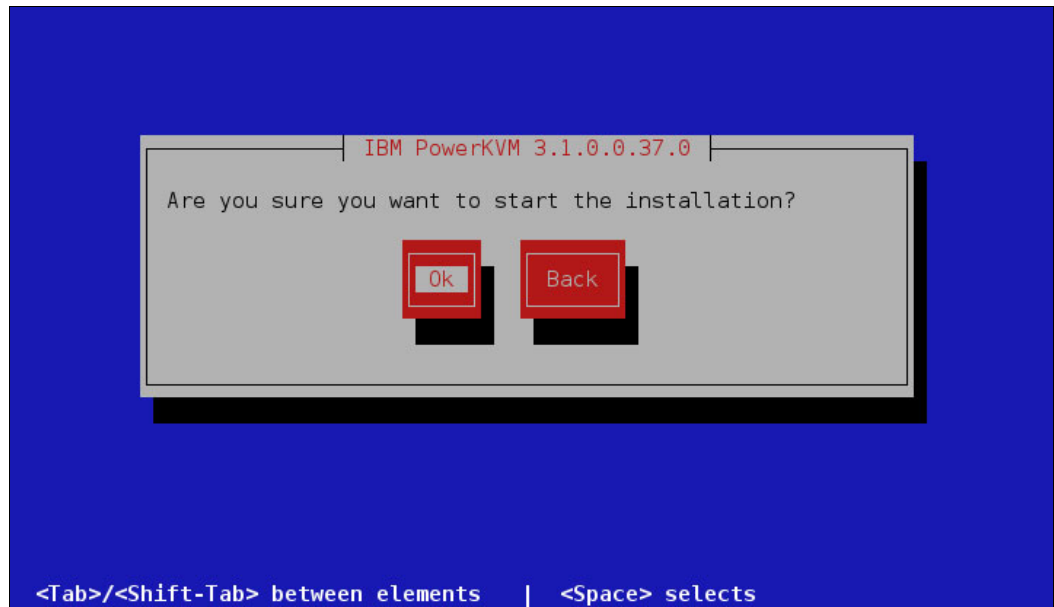


Figure 2-15 Device confirmation

18. A progress bar is displayed while installing files into the selected device. The example in Figure 2-16 shows a progress bar status of 25% complete.

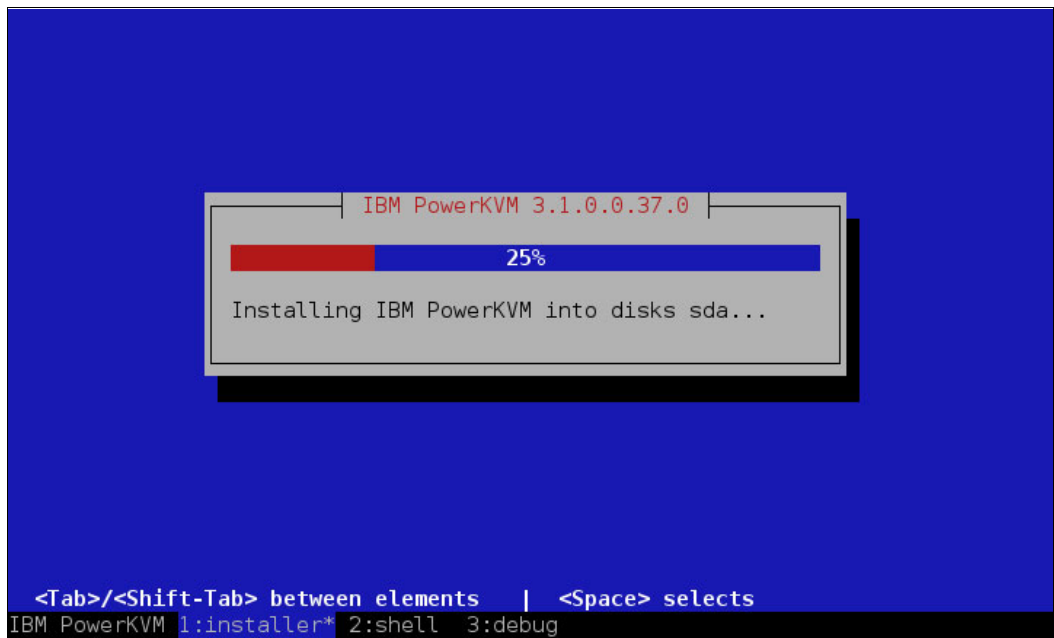


Figure 2-16 Installer progress bar

19. When the installation is finished, press Enter or the Spacebar to reboot the system.  
Figure 2-17 shows an example.

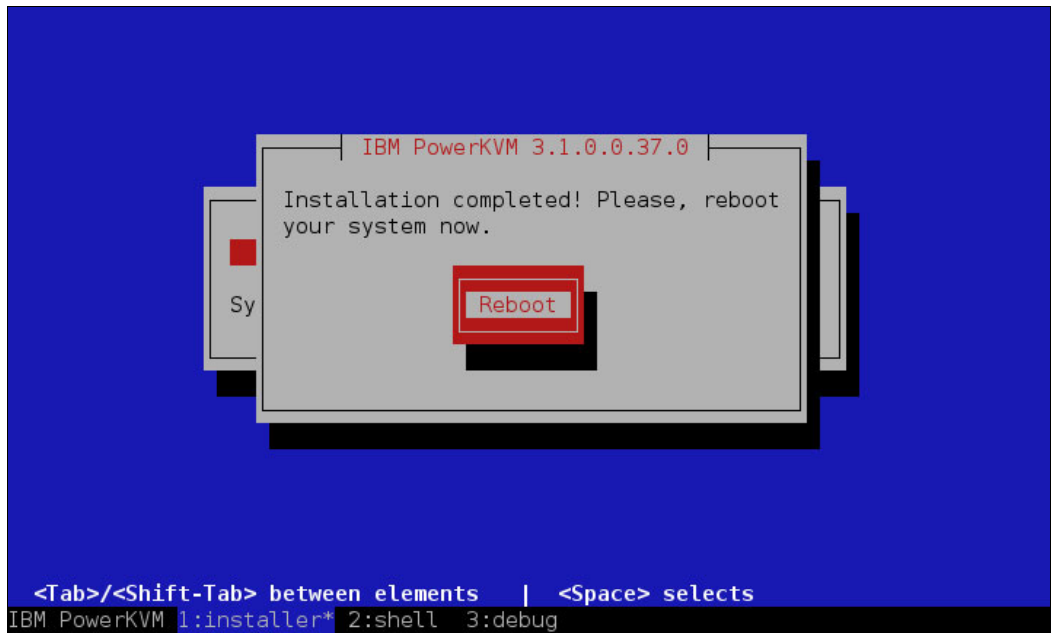


Figure 2-17 Reboot the system

20. The system reboots and automatically loads PowerKVM from the installed device, as shown in Figure 2-18.

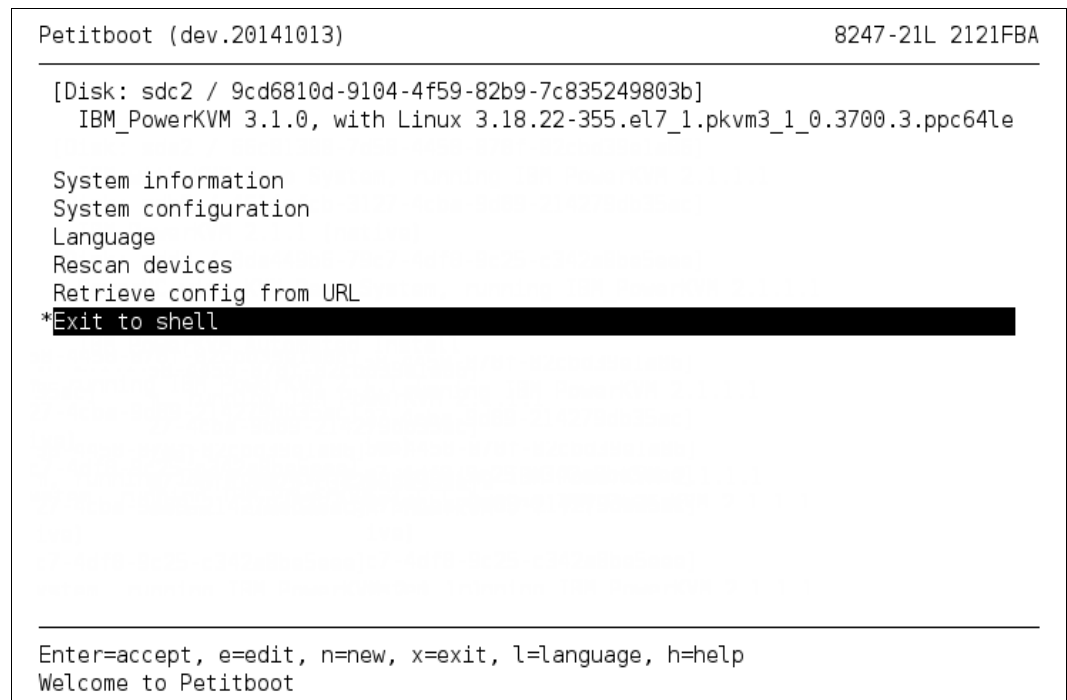


Figure 2-18 PowerKVM is automatically loaded from the installed device after reboot

## 2.2 Boot over network

This section guides you through the steps needed to boot the PowerKVM image over the network (also referred to as *netboot*) and it describes how to automate installations. The netboot process requires you to configure network services, such as DHCP, HTTP (or FTP), and TFTP, on another system. In the following examples, we used a Linux system to configure the services.

The following are the requirements for preparing the PowerKVM netboot process:

- ▶ PowerKVM ISO image
- ▶ DHCP server in the same subnet as the target machine
- ▶ HTTP (or FTP) and TFTP server that can be reached from the target machine. From now on, we refer to this server as the *netboot server*

To perform an automated installation, perform the following steps:

1. Obtain a PowerKVM ISO image, as described in 1.2.3, “Where to download PowerKVM” on page 12, and upload it to your netboot server.
2. Set up a DHCP server, as described in “Automated boot over DHCP” on page 46.
3. Set up a TFTP server, as described in “Configuration of a TFTP server” on page 47.
4. Create a boot configuration file for the target machine. A sample file is shown in Example 2-5 on page 48.
5. Set up an HTTP (or FTP) service on the netboot server, as described in “Configuration of an HTTP server” on page 46.
6. Power on the PowerKVM server.
7. Access the IPMI console of the target machine.
8. Watch the boot messages on the IPMI console and wait for the Petitboot menu to be presented with your boot entry.

## 2.2.1 Retrieve configuration from URL

Petitboot allows you to download a boot configuration file, `pxe.conf`, from a remote HTTP server.

1. On the Petitboot menu, select **Retrieve config from URL**, as shown in Figure 2-19.

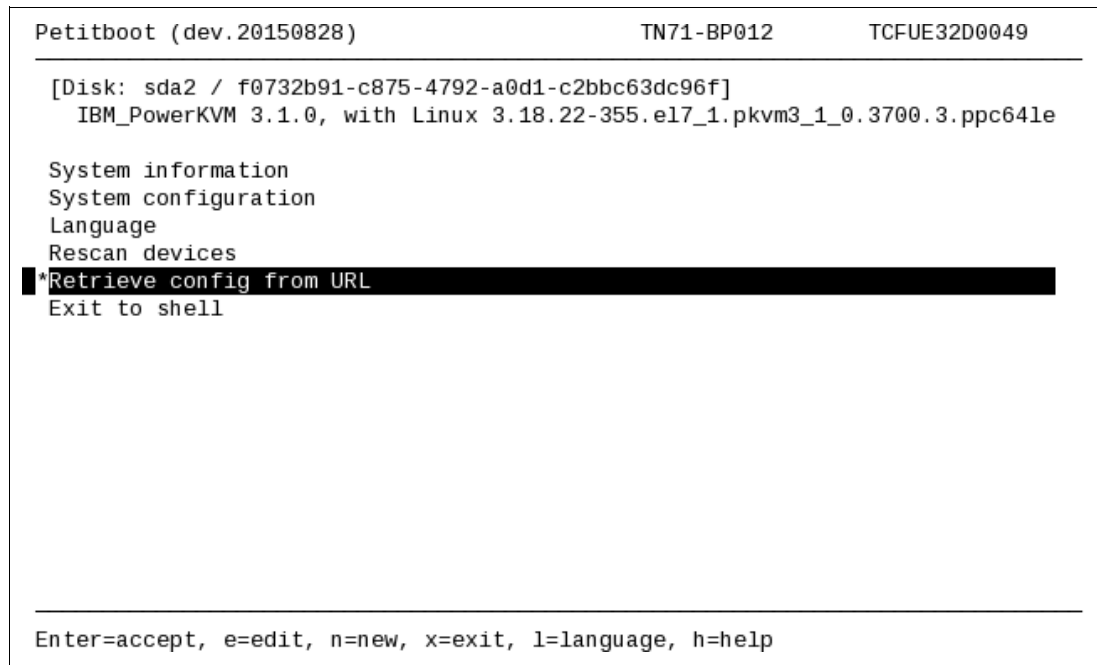


Figure 2-19 Petitboot Retrieve config from URL entry

2. You see the **Petitboot Config Retrieval** window, as shown in Figure 2-20. Enter your remote location in **Configuration URL**, and select **OK**.

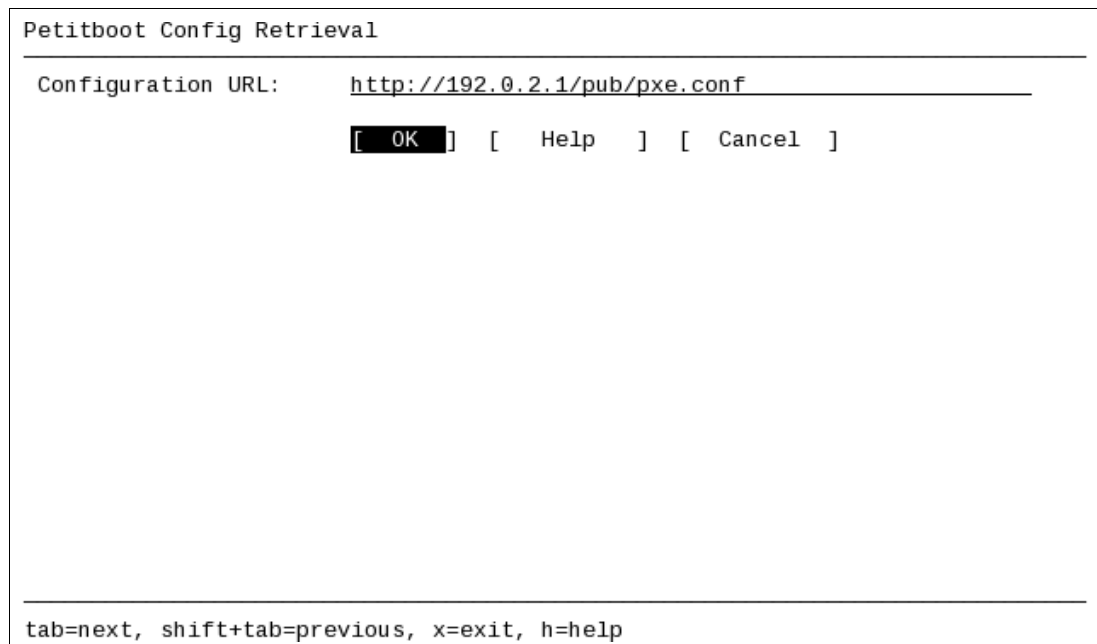


Figure 2-20 Petitboot Config Retrieval window

3. You see your entry listed in the Petitboot menu, as shown in Figure 2-21.

```

Petitboot (dev.20150828)                               TN71-BP012          TCFUE32D0049
-----
[Disk: sda2 / f0732b91-c875-4792-a0d1-c2bbc63dc96f]
  IBM_PowerKVM 3.1.0, with Linux 3.18.22-355.el7_1.pkvm3_1_0.3700.3.ppc64le
[Network: eth2 / 98:be:94:0d:f6:e6]
* Install PowerKVM 3.1.0
System information
System configuration
Language
Rescan devices
Retrieve config from URL
Exit to shell

Enter=accept, e=edit, n=new, x=exit, l=language, h=help

```

Figure 2-21 Petitboot configuration retrieved

Select **Install PowerKVM 3.1.0** and the system boots using the parameters you specified.

## 2.2.2 Boot LiveDVD over network

The following is an overview of the steps involved in booting a PowerKVM LiveDVD image over the network:

- ▶ As part of its initialization, Petitboot sends a DHCP request on all interfaces that have an active link and looks for a PXE configuration file in the DHCP response.
- ▶ If any is specified, Petitboot downloads and parses the configuration file.
- ▶ The parsed content is then displayed in the Petitboot menu.
- ▶ The menu entry is selected and PowerKVM LiveDVD is booted.
- ▶ The initial root file system is loaded from the netboot server.
- ▶ The installer is loaded and packages can be installed from a remote repository.

The netboot process can load files from the network by using different protocols (Table 2-1).

Table 2-1 Netboot supported protocols

Netboot action	Configuration	Supported protocols
Read pxe.conf	option conf-file <i>location</i>	HTTP, FTP, and TFTP
Load kernel and initrd	kernel <i>location</i> initrd <i>location</i>	HTTP, FTP, and TFTP
Load minimal root file system	root=live: <i>location</i>	HTTP and FTP (compressed read-only file system image); NFS (ISO image)
Installation repository	repo= <i>location</i>	HTTP and FTP

An example of the boot configuration file, `pxe.conf`, is shown in Example 2-1.

*Example 2-1 Petitboot configuration file*

---

```
label Your_label
kernel http://server-address/path/vmlinuz
initrd ftp://server-address/path/initrd.img
append root=live:http://server-address/path/squashfs.imgrd.dm=0 rd.md=0 \
    repo=ftp://server-address/path console=hvc0 console=tty0
```

---

The sections ahead describe how to configure the network services HTTP, DHCP, and TFTP on the netboot server.

**Note:** You must set file permissions and firewall rules to allow incoming requests to the DHCP, HTTP, or FTP services that you are configuring. These settings depend on the Linux distribution that you are running.

### 2.2.3 Automated boot over DHCP

The boot over DHCP is commonly used to boot PowerKVM images over the network. This is a common practice in automated environments.

**Note:** The DHCP server needs to be configured in the same subnetwork that the target system boots. Otherwise, the target system will not be able to discover DHCP, and boot through PXE will not work.

1. Configure the DHCP server to be capable of reading text files from TFTP server using the `conf-file` option. Example 2-2 shows a sample configuration.

*Example 2-2 DHCP configuration sample*

---

```
ddns-update-style none;
option conf-file code 209 = text;
subnet 192.0.2.0 netmask 255.255.255.0 {
    option routers 192.0.2.1;
    option domain-name-servers 192.0.2.1, 192.0.2.2;
    host powerkvm-host {
        hardware ethernet 6c:ae:8b:01:a0:f4;
        fixed-address 192.0.2.10;
        option host-name "powerkvm-host";
        option conf-file "http://192.0.2.1/powerkvm/pxe.conf";
    }
}
```

---

2. Restart DHCP service to apply changes.

#### Configuration of an HTTP server

The HTTP server needs to provide the following files and directory over its `wwwroot` directory:

- ▶ `vmlinuz` (kernel)
- ▶ `initrd.img` (initrd)
- ▶ `squashfs.img` (compressed read-only root file system)
- ▶ `repodata/` (the directory containing the repository metadata)

To serve the netboot files through HTTP, follow these steps:

1. Extract the PowerKVM ISO image under the `wwwroot` directory of your HTTP server. You will have a similar file structure as follows:

```
boot/  
etc/  
LiveOS/  
packages/  
ppc/  
TRANS.TBL  
VERSION  
.discinfo
```

2. Update your boot configuration file, `pxe.conf`, to point to the correct locations where `mlinuz`, `initrd.img`, and `squashfs.img` are. Also, set the `repo` parameter to point to the correct repository used during installation. In Example 2-3, the files extracted from the PowerKVM ISO image are under the `wwwroot/powerkvm` directory.

*Example 2-3 Boot configuration sample*

---

```
label PowerKVM NetBoot Install  
kernel http://192.0.2.1/powerkvm/ppc/ppc64le/vmlinuz  
initrd http://192.0.2.1/powerkvm/ppc/ppc64le/initrd.img  
append root=live:http://192.0.2.1/powerkvm/LiveOS/squashfs.img rd.dm=0 rd.md=0  
repo=http://192.0.2.1/powerkvm/packages console=hvc0 console=tty0
```

---

## Configuration of a TFTP server

It is also possible to provide a `pxe.conf` through TFTP protocol instead of HTTP. Just enable the TFTP service and copy the `pxe.conf` file, which you set in the DHCP configuration file, to the TFTP root directory.

1. If your TFTP server is controlled by `xinetd` service, the `/etc/xinetd.d/tftp` configuration file will look like in Example 2-4.

*Example 2-4 TFTP configuration sample*

---

```
service tftp  
{  
    socket_type      = dgram  
    protocol         = udp  
    wait             = yes  
    user             = root  
    server            = /usr/sbin/in.tftpd  
    server_args       = -B 65464 -v -s /srv/netboot  
    disable           = no  
    per_source        = 11  
    cps               = 100 2  
    flags             = IPv4  
}
```

---

In the prior example, `pxe.conf` file is placed in the `/srv/netboot` directory.

2. Ensure that you also update the line `option conf-file` inside the host block of your DHCP configuration file to look like the following:

```
...
host powerkvm-host {
    hardware ethernet 6c:ae:8b:01:a0:f4;
    fixed-address 192.0.2.10;option host-name "powerkvm-host";
    option conf-file "pxe.conf";
}
...
```

3. Restart the TFTP and DHCP services to get the changes applied.

## 2.2.4 Automated installation

A PowerKVM installation can be performed unattended by providing a kickstart-like file to the boot parameters, the `kvm.inst.auto` option in the boot configuration file, or the Petitboot menu entry.

The automated installation requires these elements:

- ▶ `kvm.inst.auto=location` in the boot configuration file
- ▶ Kickstart-like file to be downloaded by the installer

**Note:** Automatic installation can be executed only on the `hvc0` console. That is the console type supported by `ipmitool` and QEMU with the `-nographic` option.

The boot option `kvm.inst.auto` supports HTTP, TFTP, and NFS protocols.

To download the kickstart file specified in the `kvm.inst.auto` option, the installer needs to have network access during boot. The network settings can be specified by using the following parameters in the boot configuration file:

- ▶ `ifname=interface:mac`
- ▶ `ip=ip:server-id:gateway:netmask:hostname:interface:none`
- ▶ `ip=interface:dhcp`

The `server-id` is rarely used so it is common to omit its value, for example:

```
ip=192.0.2.10::192.0.2.254:255.255.255.0:powerkvm-host:net0:none
```

If you specify a domain name as the kickstart file location, the `kvm.inst.auto` option, for example `server.example.com/pub/powerkvm.ks`, instead of an IP address, you will need to specify the name servers to be used to resolve `server.example.com` into an IP address:

- ▶ `nameserver=server`

The `nameserver` parameter can be specified up to three times. The exceeding appearances of this parameter will be ignored.

The following Example 2-5 shows the content of a sample Petitboot configuration file with network settings and a kickstart file specified to perform an unattended installation.

*Example 2-5 Petitboot configuration sample for unattended installation*

```
label IBM PowerKVM Automated Install (kickstart)
kernel http://192.0.2.1/kvmonp_netboot/vmlinuz
initrd http://192.0.2.1/kvmonp_netboot/initrd.img
append root=nfs:192.0.2.1:/var/www/kvmonp_netboot/iso/ibm-powerkvm.iso \
    kvm.inst.auto=tftp://192.0.2.1/kickstart/powerkvm.ks \
```



```
ifname=net0:e4:1f:13:fd:cf:7d \  
ip=192.0.2.10::192.0.2.254:255.255.255.0:powerkvm-host:net0:none \  
nameserver=192.0.2.1 rd.dm=0 rd.md=0 console=hvc0
```

---

## Supported options in a kickstart file

The following options are supported in the kickstart file that you specify in the `kvmp.inst.auto` parameter:

### ► Pre-install scripts

As the name suggests, the pre-install scripts are executed before the installation process runs. The commands need to be placed between `%pre` and `%end` tags.

```
%pre  
your shell script commands go here  
%end
```

### ► Post-install scripts

The post-install section is executed after the installation is complete and before the system is rebooted. The commands need to be placed between `%post` and `%end` tags.

```
%post  
your shell script commands go here  
%end
```

### ► Partition (required)

A very simple example of installing PowerKVM on a single disk is displayed in Example 2-6.

*Example 2-6 Kickstart sample for installing PowerKVM on a single disk*

---

```
part pv.01 --ondisk=/dev/sda  
volgroup VOL_GROUP pv.01  
logvol / --vgname=VOL_GROUP --size=100  
logvol /var/log --vgname=VOL_GROUP --size=30
```

---

In the prior example, disk `/dev/sda` is used for the installation. The LVM volume group `VOL_GROUP` is created. The LVM logical volume `/` has 100 GiB of size, and `/var/log` has 30 GiB. The logical volume `/var/lib/libvirt/images` is created automatically with the remaining space in the volume group `VOL_GROUP`.

You can also use more than one disk during installation. Just add multiple `part` lines and update the `volgroup` line in your kickstart file, as shown in Example 2-7.

*Example 2-7 Kickstart example using more than one disk for installation*

---

```
part pv.01 --ondisk=/dev/sda  
part pv.02 --ondisk=/dev/mapper/mpathb  
volgroup VOL_GROUP pv.01 pv.02  
... 
```

---

To install PowerKVM V3.1.0 on top of an existing PowerKVM V2.1.1 instance and preserve your `/var/log` and `/var/lib/libvirt/images` directories, you just need to specify the `--noformat` option to the `logvol` lines in the kickstart file, and omit the `part` and `volgroup` lines. The Example 2-8 shows how to do it.

*Example 2-8 Install PowerKVM V3.1.0 on an existing PowerKVM V2.1.1 instance*

---

```
...
logvol /var/log --vgname=ibmpkvm_vg_log --noformat
logvol /var/lib/libvirt/images --vgname=ibmpkvm_vg_data --noformat
...
```

---

► **Network** (required)

The `network` option is used to configure network settings of the target system. The settings will take place after the next reboot, when the target system will be already installed and properly configured.

**Note:** The network settings that you specify in the kickstart file are for the **target system**. They are not supposed to configure network settings of the installer LiveDVD.

Example 2-9 shows how to configure an interface using DHCP.

*Example 2-9 Configure an interface with DHCP*

---

```
network --device enp0s0 --bootproto dhcp
```

---

Example 2-10 shows how to configure an interface using a static IP address. And it also shows to specify the netmask, gateway, and name server addresses.

*Example 2-10 Configure an interface with static IP address*

---

```
network --device enp0s1 --bootproto static --ip=10.34.102.222
--netmask=255.255.255.0 --gateway=10.34.102.254 --nameserver=10.34.102.1
--nameserver=10.34.102.2
```

---

► **Root password**

The password of user `root` can be in plain text or encrypted.

The following example shows how to specify the password in plain text (not encrypted):

```
rootpw plain-text-password
```

And the following example shows how to specify an encrypted password:

```
rootpw --iscrypted password-hash-in-sha512-format
```

You can generate a hash of your password by executing the following shell command:

```
echo -n "Your password" | sha512sum
```

Or by using the module `hashlib` from the Python standard library:

```
python -c "import hashlib; print(hashlib.sha512(b'Your password')).hexdigest()"
```

► **Time zone**

With the `timezone` option, you can define any of the time zones available under the `/usr/share/zoneinfo` directory. For example:

```
timezone America/Sao_Paulo
```

If `--utc` option is specified, the system assumes the hardware clock is set to UTC (Coordinated Universal Time).

You can specify the `--nntp` option to disable automatic starting of NTP service.

You can also specify a comma-separated list of NTP servers, using the `--ntpserver`s option, to keep the system time up to date. The following example shows how to do it.

```
timezone --utc --ntpserver=server1,server2 America/Sao_Paulo
```

► **Language**

The `lang` option is used to specify the language of the target system. For example:

```
lang language
```

The supported languages are: `de_DE`, `en_US`, `es_ES`, `fr_FR`, `it_IT`, `Ja_JP`, `ko_KR`, `pt_BR`, `ru_RU`, `zh_CN`, `zh_TW`.

A sample kickstart file is shown in Example 2-11.

*Example 2-11 Kickstart file sample*

---

```
%pre
your shell commands to be executed before installation
%end

part pv.01 --ondisk=/dev/sda
part pv.02 --ondisk=/dev/sdb
volgroup PKVM_SYS_VG pv.01 pv.02
logvol / --vgname=PKVM_SYS_VG --size=50
logvol /var/log --vgname=PKVM_SYS_VG --size=20

network --device=net0 --bootproto=static --ip=192.0.2.10 --netmask=255.255.255.0
--gateway=192.0.2.1 --nameserver=192.0.2.1

rootpw topsecret

timezone --utc --ntpserver=0.us.pool.ntp.org America/Sao_Paulo
lang pt_BR

%post
your shell commands to be executed after installation
%end
```

---

On the installed system, you can find the files from the automated installation:

- Log files: `/var/log/powerkvm/*.log`

- Kickstart file: `/var/log/powerkvm/kop.ks`
- Pre and post install scripts
  - `/var/log/powerkvm/preScriptKS.sh`: Contains the commands specified in the `%pre` section of the kickstart file. If the `%pre` section was not specified, this file will not exist on the installed system.
  - `/var/log/powerkvm/preScriptKS.log`: The standard output of the `%pre` section commands.
  - `/var/log/powerkvm/preScriptKS.err`: The standard error messages of the `%pre` section commands.
  - `/var/log/powerkvm/postScriptKS.sh`: Contains the commands specified in the `%post` section of the kickstart file.
  - `/var/log/powerkvm/postScriptKS.log`: The standard output of the `%post` section commands.
  - `/var/log/powerkvm/postScriptKS.err`: The standard error messages of the `%post` section commands.

## Rename the network interface

By default, the interface names are defined by the **Predictable Network Interface Names** feature from the `systemd` program. More details about this feature can be found here:

<https://wiki.freedesktop.org/www/Software/systemd/PredictableNetworkInterfaceNames>

You can change the predictable name by using the `ifname` option in the boot configuration file, `pxe.conf`. Typically, you just need to specify the hardware address of the interface to be renamed, for example:

```
ifname=new-nic-name:mac-address
```

Suppose that your interface hardware address is `00:11:22:33:44:55` and you want to refer to it as `lan0`. Just add the following to your boot line arguments:

```
ifname=lan0:00:11:22:33:44:55 ip=lan0:dhcp
```

You can also refer to `lan0` in the kickstart file:

```
network --device lan0 --bootproto dhcp
```

After the system is installed, the interface `lan0` exists with the configuration that you specified in the kickstart file.

## 2.3 Install over existing IBM PowerKVM and host migration

This section describes how to reinstall and migrate an existing IBM PowerKVM system to the version 3.1.0. The installed host system can be running a PowerKVM V2.1.1.3 or later release.

In case of an existing 3.1.0 instance, the reinstallation can be used as a rescue method for reinstalling the host without losing the guest disk images.

In case of a previous 2.1.1.x instance, the reinstallation is considered a host migration. Before proceeding with the migration, understand its limitations and make sure the following requirements are satisfied:

- ▶ The host needs to be running PowerKVM version 2.1.1.3 or later. Older versions are not supported.
- ▶ Downgrade migration is not supported. That means you cannot migrate from version 3.1.0 to version 2.1.1.3.

It is a preferred practice to save the current libvirt configuration files so you can be able to convert them to the new syntax and have your guests up and running after the host is completely migrated. To save libvirt configuration files, you can use the following commands:

```
mkdir -p /var/lib/libvirt/images/backup
tar cvzf /var/lib/libvirt/images/backup/libvirt-xml-files.tar.gz /etc/libvirt
```

After that, place `libvirt-xml-files.tar.gz` in a safe storage media and you are ready to start the migration process.

In the installation process, when the installer detects a previous instance of PowerKVM, it offers the option **Install over existing IBM PowerKVM**, as shown in Figure 2-22. The guest images on the *data* partition are preserved. Only the *root*, *swap*, and *boot* partitions of the host are erased and reinstalled.

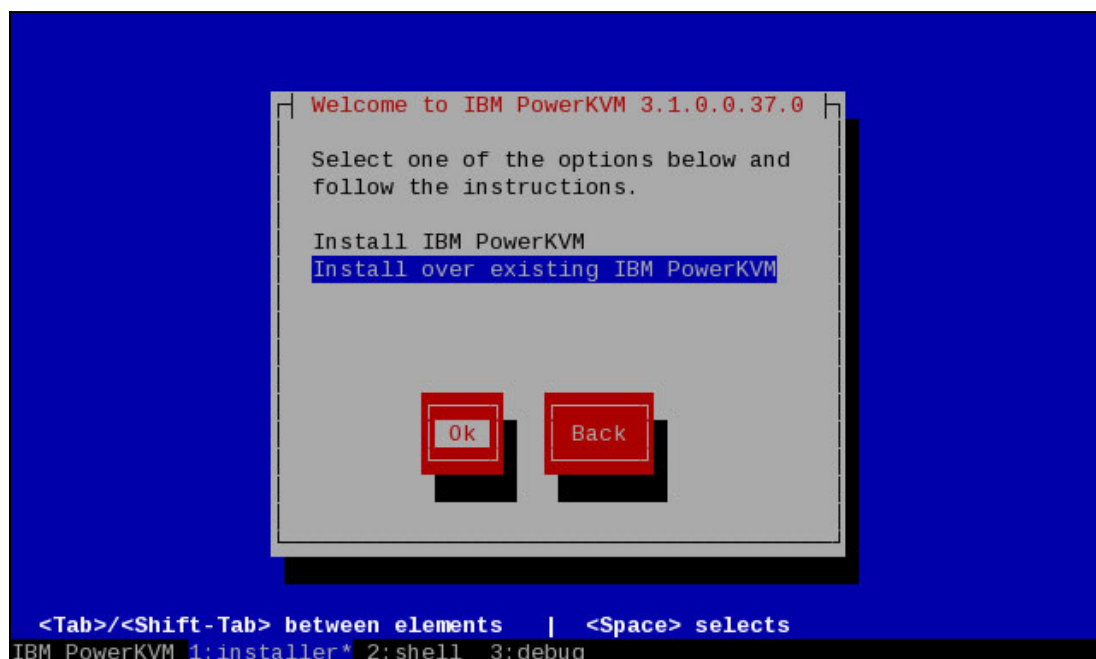


Figure 2-22 Install over existing IBM PowerKVM option displayed in the installer menu

After selecting **Install over existing IBM PowerKVM**, you are prompted to select the volume group. Choose **ibmpkvm\_vg\_root** and select **OK**, as shown in Figure 2-23.

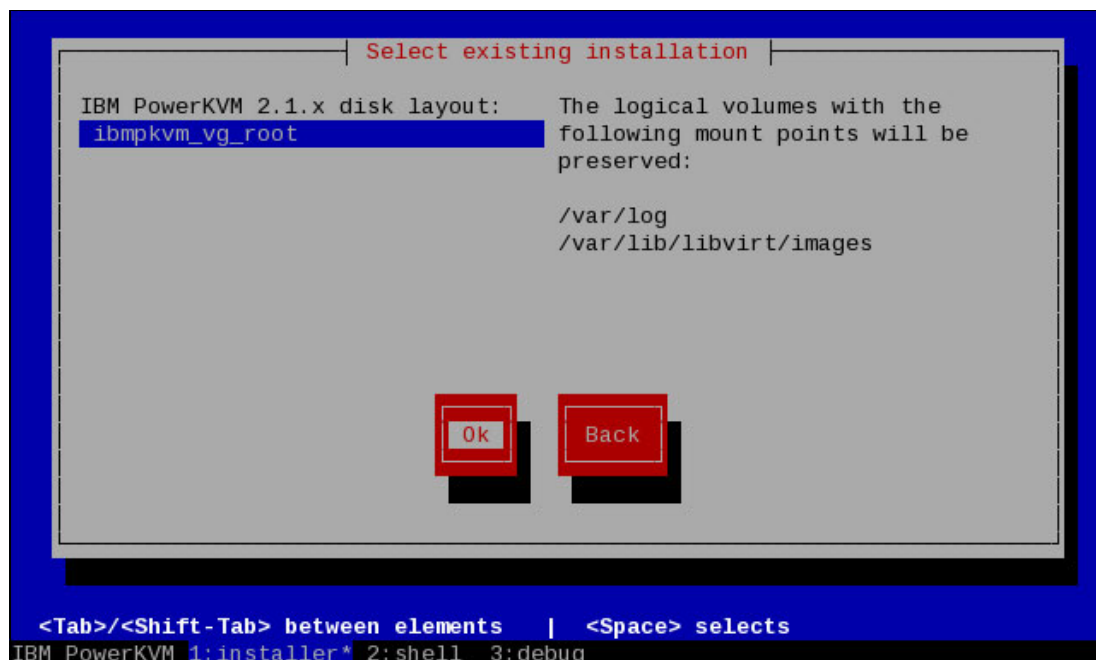


Figure 2-23 Select existing installation window

After this point, the installer steps are much the same as described in section 2.1, “Host installation” on page 32.

After the installation of PowerKVM is completed, you can convert the libvirt configuration files to the new syntax used in PowerKVM V3.1.0. The following steps can be used only to convert libvirt configuration files syntax from PowerKVM version 2.1.1.3 to 3.1.0. If you were already running version 3.1.0, you can skip the following steps:

1. Install the `powerkvm-xml-toolkit` package. If it is not installed by default, use the following command:  

```
yum install powerkvm-xml-toolkit
```
2. Extract the saved libvirt configuration files by running:  

```
tar xvf /var/lib/libvirt/images/backup/libvirt-xml-files.tar.gz -C /
```
3. Run the `xml-toolkit.sh` script to convert all libvirt configuration files. The following example shows how it can be done.

*Example 2-12 Convert libvirt configuration files using powerkvm-xml-tool*

---

```
xmlfiles=$(find /etc/libvirt/qemu /etc/libvirt/qemu/networks \
    /etc/libvirt/storage -type f -name "*.xml")
for f in $xmlfiles; do
    xml-toolkit.sh -i $f
done
```

---

For each converted file, a backup file is created with the `.bak` extension. For additional help, run `xml-toolkit.sh -h`.

4. Also, ensure that your network and storage settings are configured to start automatically when `libvirtd` service initiates. You can guarantee this by creating a symbolic link in the autostart directories, as the following:

```
ln -s ../default.xml /etc/libvirt/qemu/networks/autostart/default.xml
ln -s ../default.xml /etc/libvirt/storage/autostart/default.xml
```

5. Restart `libvirtd` and `kimchid` services by running:

```
systemctl restart libvirtd
systemctl restart kimchid
```

At this point, your host and guests are migrated to the new PowerKVM version 3.1.0.

## 2.4 Configuration

This section describes how to use the `ibm-configure-system` tool that is included in the PowerKVM installation to change some important settings of the machine.

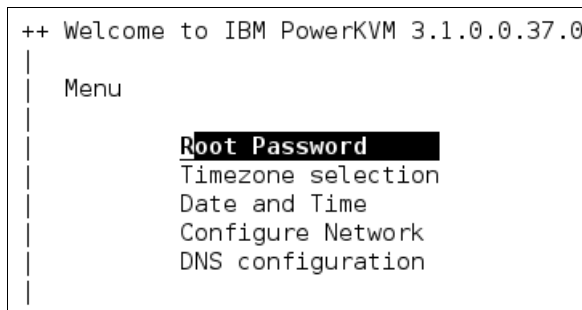
You can use the `ibm-configure-system` tool to perform the following maintenance tasks on the installed system:

- ▶ Reset the root password
- ▶ Select the time zone
- ▶ Set the date and time
- ▶ Configure the network
- ▶ Configure the DNS

To execute the configuration tool, from the root shell, run this command:

```
# ibm-configure-system
```

The `ibm-configure-system` main panel is shown in Figure 2-24.



```
++ Welcome to IBM PowerKVM 3.1.0.0.37.0
|
|  Menu
|
|  Root Password
|  Timezone selection
|  Date and Time
|  Configure Network
|  DNS configuration
|
```

Figure 2-24 The `ibm-configure-system` tool

## Reset the password

The root password can be changed from the `ibm-configure-system` tool by selecting the **Root Password** option from the menu. The password reset is shown in Figure 2-25.

```
+-----+ Root Password +-----+
|
|  Pick a root password. You must type it
|  twice to ensure you know what it is and
|  didn't make a mistake in typing. Remember
|  that the root password is a critical part
|  of system security!
|
|  Password:          *
|  Password (confirm): *
|
|          +-----+ +-----+
|          | OK |   | Back |
|          +-----+ +-----+
|
+-----+
```

Figure 2-25 Reset password

## Timezone selection

By selecting **Timezone selection** in the configuration-system tool, you can adjust the time zone as shown in Figure 2-26.

```
+-----+ Timezone Selection +-----+
|
|  Select the timezone for the system
|
|  Europe/Andorra      ^
|  Europe/Athens       :
|  Europe/Belgrade     #
|  Europe/Berlin       :
|  Europe/Bratislava   v
|
|  [ ] System clock uses UTC
|
|          +-----+ +-----+
|          | OK |   | Back |
|          +-----+ +-----+
|
+-----+
```

Figure 2-26 Timezone selection

## Configure the network

You can change the network settings on a persistent basis by selecting the **Configure Network** option from `ibm-configure-system` tool.

**Tip:** Use the `ibm-configure-system` tool if the network is not yet configured. All of the network scripts and settings are automatically changed in the system.



The device list is shown in Figure 2-27. Select the device that you want to configure and click **OK**. The configuration details for the selected device are shown in Figure 2-28.

```

+-----+ Configure network +-----+
|
| Select the device on the list to be configured:
|   enP2pls0f0 [ 00:0e:1e:80:60:84 ] (DOWN)  ^
|   enP2pls0f1 [ 00:0e:1e:80:60:85 ] (DOWN)  :
|   enP2pls0f2 [ 00:0e:1e:80:60:86 ] (DOWN)  #
|   enP2pls0f3 [ 00:0e:1e:80:60:87 ] (DOWN)  :
|   enP3p9s0f0 [ 40:f2:e9:5d:4f:e0 ] (UP)    v
|
|   +-----+ +-----+
|   | OK |   | Back |
|   +-----+ +-----+
|
+-----+

```

Figure 2-27 Select network device

```

+-----+ Network Device Configuration +-----+
|
| Device                               enP3p9s0f0
| MAC Address                         40:f2:e9:5d:4f:e0
| Use DHCP                           [ ]
| Static IP                           10.150.60.56
| Netmask                             255.255.0.0
| Default gateway IP                  10.150.254.32
| Create Network Bridge (brenP3p9s0f0) [ ]
| Enable                              [*]
|
|   +-----+ +-----+
|   | Save |   | Back |
|   +-----+ +-----+
|
+-----+

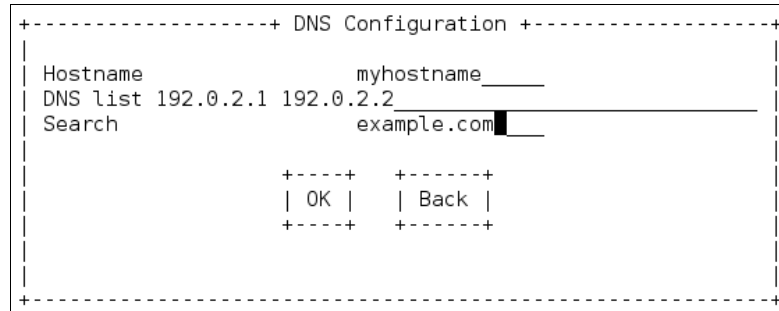
```

Figure 2-28 Network device configuration

**Note:** The network can be also changed by editing the `ifcfg-*` files under `/etc/sysconfig/network-scripts` followed by an `ifdown` and `ifup` command for the changed interface.

## DNS configuration

If you want to change the DNS configuration, select the **DNS configuration** option from the `ibm-configure-system` tool and specify the host name, the IP list of DNS servers, and the search domain, as shown in Figure 2-29.



```
+-----+ DNS Configuration +-----+
|
| Hostname          myhostname_____
| DNS list 192.0.2.1 192.0.2.2_____
| Search           example.com_____
|
|               +-----+ +-----+
|               | OK |   | Back |
|               +-----+ +-----+
|
+-----+
```

Figure 2-29 DNS configuration change

### 2.4.1 Console configuration for Scale-out Power Systems

PowerKVM supports serial and SoL console types. This section shows how to set an IPMI (SoL) console password on the FSP by using the ASM interface.

To configure an IPMI console password, follow these steps:

1. Ensure that OPAL is selected as the Hypervisor Mode. To change the Hypervisor Mode, first the system needs to be powered off. Then, log in to the ASM interface and expand the **System Configuration** menu. Click **Firmware Configuration**, and change the firmware type to OPAL, as shown in Figure 2-30.

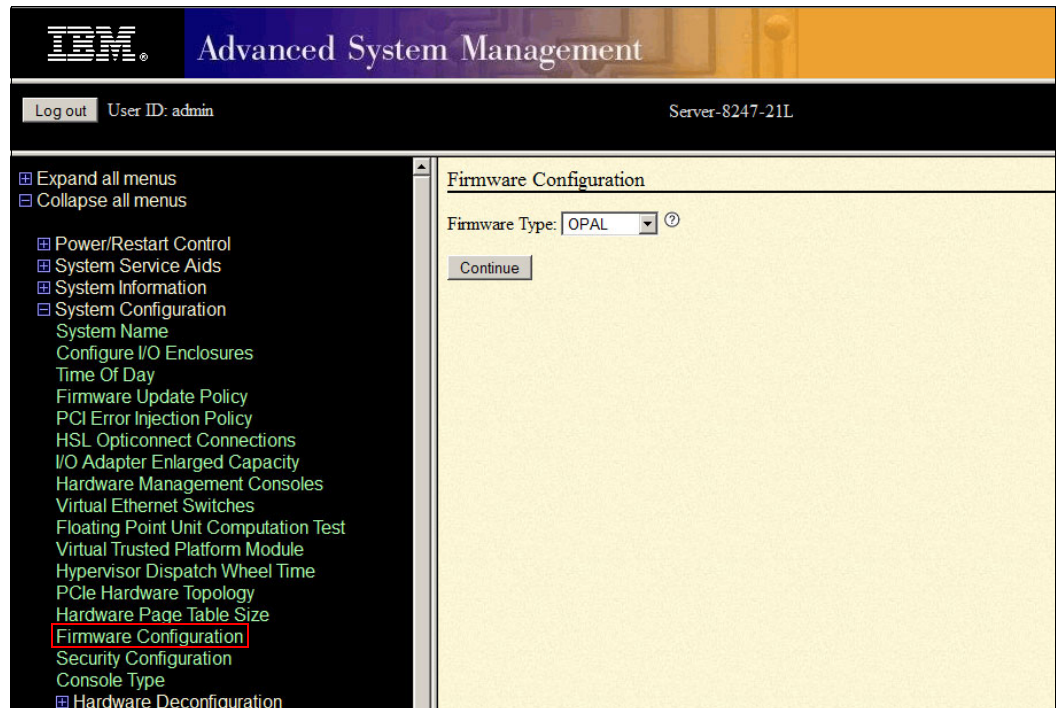


Figure 2-30 Select firmware configuration

2. To set or change the IPMI password, click **Login Profile**, and then **Change Password**. There, select IPMI as the **User ID to change**. If an old password was set, enter it as the current password and set the new password by entering it twice, as shown in Figure 2-31.

The screenshot shows the IBM Advanced System Management web interface. The top header includes the IBM logo and the title 'Advanced System Management'. Below the header, there's a navigation bar with 'Log out' and 'User ID: admin'. The server identifier 'Server-8247-21L-SN2121FBA' is displayed on the right. The left sidebar contains a tree of options: 'Expand all menus', 'Collapse all menus', 'Power/Restart Control', 'System Service Aids', 'System Information', 'System Configuration', 'Network Services', 'Performance Setup', 'On Demand Utilities', 'Concurrent Maintenance', 'Login Profile', 'Change Password' (highlighted with a red box), 'Retrieve Login Audits', 'Change Default Language', 'Update Installed Languages', and 'User Access Policy'. The main content area is titled 'Change Password' and contains the following fields: 'User ID to change:' with a dropdown menu showing 'IPMI' and a help icon; 'Current password for user ID admin:' with a password input field and a help icon; 'New password for user:' with a password input field and a help icon; and 'New password again:' with a password input field and a help icon. A 'Continue' button is located at the bottom of the form.

Figure 2-31 Changing the IPMI password

**Note:** To enable the IPMI console for the first time, reset the service processor after setting the IPMI console password. Expand **System Service Aids** on the left panel and click **Reset Service Processor**.

## 2.4.2 Console configuration for Power LC systems

The new Power LC Systems use OPAL as the system firmware and a baseboard management controller (BMC) as service processor and control interface to the user. BMC supports the Intelligent Platform Management Interface (IPMI 2.0) and Data Center Management Interface (DCMI 1.5) for system monitoring and management.

To connect to the server the first time you have two possibilities:

1. Connect with IPMI using an IP address issued by a DHCP server

DHCP is the default network set-up for Power LC Systems. If you are using a DHCP server and know the IP address that your system was assigned, continue powering on your system with IPMI as described in chapter 2.1, "Host installation" on page 32.

If you do not know the IP address or plan to use a static IP address, you must connect to your system by using a serial console session or using an ASCII terminal.

2. Connect using a serial console

If you are using a serial console, follow these steps:

- a. Attach the Serial to RJ-45 cable to serial port on Power system.
- b. Attach USB connection to USB port on either PC or notebook.
- c. Open a terminal emulator program such as PuTTY or minicom.

- d. Set the communications to use the following options:
    - i. 115200 baud rate
    - ii. Data bits of 8
    - iii. Parity of None
    - iv. Stop bits of 1
  - e. Power on your server by using the power button on the front of your system. Your system powers on to the Petitboot bootloader menu. This process takes about 1 - 2 minutes to complete. Do not walk away from your system. When Petitboot loads, your monitor or serial connection becomes active and you will need to push any key in order to interrupt the boot process.
  - f. At the Petitboot bootloader main menu, select **Exit to Shell**.
  - g. Run **ipmitool lan print 1**. If this command returns an IP address, you can continue using that address to install PowerKVM as described in chapter 2.1, "Host installation" on page 32.
- If no IP addresses are returned, follow these steps:
- a. Set the mode to static by running this command: **ipmitool lan set 1 ipsrc static**.
  - b. Set your IP address by running this command: **ipmitool lan set 1 ipaddr ip\_address** where **ip\_address** is the static IP address that you are assigning to this system.
  - c. Set your netmask by running this command: **ipmitool lan set 1 netmask netmask\_address** where **netmask\_address** is the netmask for the system.
  - d. Set your gateway server by running this command: **ipmitool lan set 1 defgw ipaddr gateway\_server** where **gateway\_server** is the gateway for this system.
  - e. Confirm the IP address by running the **ipmitool lan print 1** command again.
  - f. Power your system off by running the **ipmitool power off** command.
  - g. If you set a static IP address in step b, unplug the power cords from the back of the system. Wait 30 seconds and then apply power to boot the BMC.



## Managing hosts and guests from a Web interface

This chapter describes how to manage hosts and guest systems by using a web interface.

First, we introduce *Kimchi*, a web-based management tool. Kimchi isolates the administrator from the task of remembering command-line syntax. But, as seasoned administrators know, sometimes it is best to study the commands and learn their capability to better understand what happens in the graphical user interface (GUI). For that, we recommend a quick scan of Chapter 4, “Managing guests from the command-line interface” on page 105.

Host and guest management include the following tasks:

- ▶ Manage host networks
- ▶ Manage guest networks
- ▶ Manage storage
- ▶ Manage network
- ▶ Creating templates for guest management
- ▶ Using templates for guest management
- ▶ Accessing a graphical interface
- ▶ Administration through Ginger

## 3.1 Kimchi

Kimchi is an open source web-based management tool for IBM PowerKVM virtual machines. This software is installed and configured in a PowerKVM system by default.

Kimchi includes these features, among others:

- ▶ Easy to use HTML5 interface
- ▶ Templates to speed up guest creation
- ▶ Guest installation using remote ISO
- ▶ Web Virtual Network Computing (VNC) tool to access guests
- ▶ Storage pools
- ▶ NAT and bridge networks
- ▶ Host information and statistics
- ▶ Host administration tool (Ginger)

Several improvements were made in Kimchi for this new PowerKVM release:

- ▶ Virtual NIC hot-plug support
- ▶ Upload file to storage pool
- ▶ Make template defaults configurable through a file
- ▶ Guest pause/resume support
- ▶ Support to edit guest MAC address
- ▶ Allow user changes guest disk format on template level
- ▶ Create guests asynchronously
- ▶ Bugfixes

### 3.1.1 Accessing Kimchi

An HTML compatible browser is necessary to access Kimchi. The latest Firefox extended support release (ESR) is recommended, but the latest versions of the following browsers should work as well:

- ▶ Chrome
- ▶ Internet Explorer
- ▶ Opera
- ▶ Safari
- ▶ Safari iOS
- ▶ Android Browser

**Note:** To access Kimchi, use the `https://IPADDRESS:8001/` URL, where *IPADDRESS* is the IP address that you configured for the server during installation. For example, if you used 192.0.2.10, the URL is `https://192.0.2.10:8001/`.

Kimchi uses Pluggable Authentication Modules (PAMs) for user authentication. It means that any user account, registered on the host, is able to access Kimchi including the root account.

Users with no administration privileges logged in Kimchi are allowed to read only. However, the administrator can authorize users to access their guests. Read 3.6.2, “Guest management” on page 86 to know how to configure a guest.

**Note:** We suggest avoiding the use of the root account on Kimchi and creating specific accounts that provide the needed administration privileges.

Figure 3-1 shows the **Login** panel. Type the user name, the password, then click **Log in**.

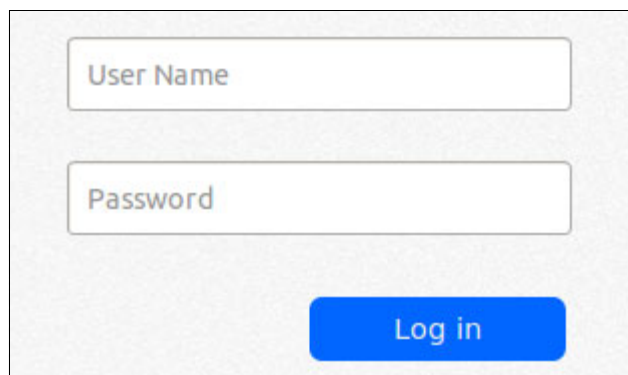
The image shows a login panel with a light gray background. It contains two text input fields: the top one is labeled "User Name" and the bottom one is labeled "Password". Below these fields is a blue button with the text "Log in" in white.

Figure 3-1 Login panel

### 3.1.2 Getting help

Figure 3-2 shows how to access the help menu. Click user name at upper right for the help menu. Click **Help** to open a new window with information about how to use Kimchi.

Using the same menu, it is possible to find out the installed Kimchi version and to safely log out of Kimchi.

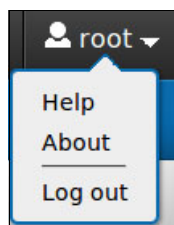


Figure 3-2 Accessing help

## 3.2 Host tab

The first tab displayed in Kimchi is named *Host*. It displays basic information about the host system and it offers a basic host management. Figure 3-3 shows all tabs existing in Kimchi.

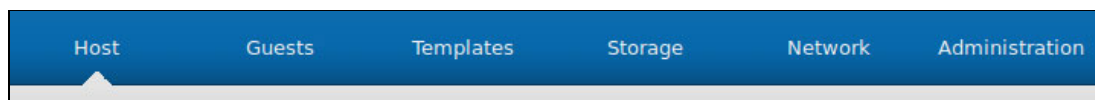


Figure 3-3 Host tab selected

Figure 3-4 shows the Host tab. Users can shut down, restart, or connect to the host by using VNC. It is also possible to have quick information about the system, such as:

- ▶ OS Distro name
- ▶ OS Version
- ▶ OS Code Name
- ▶ Processor
- ▶ Number of CPUs
- ▶ Memory

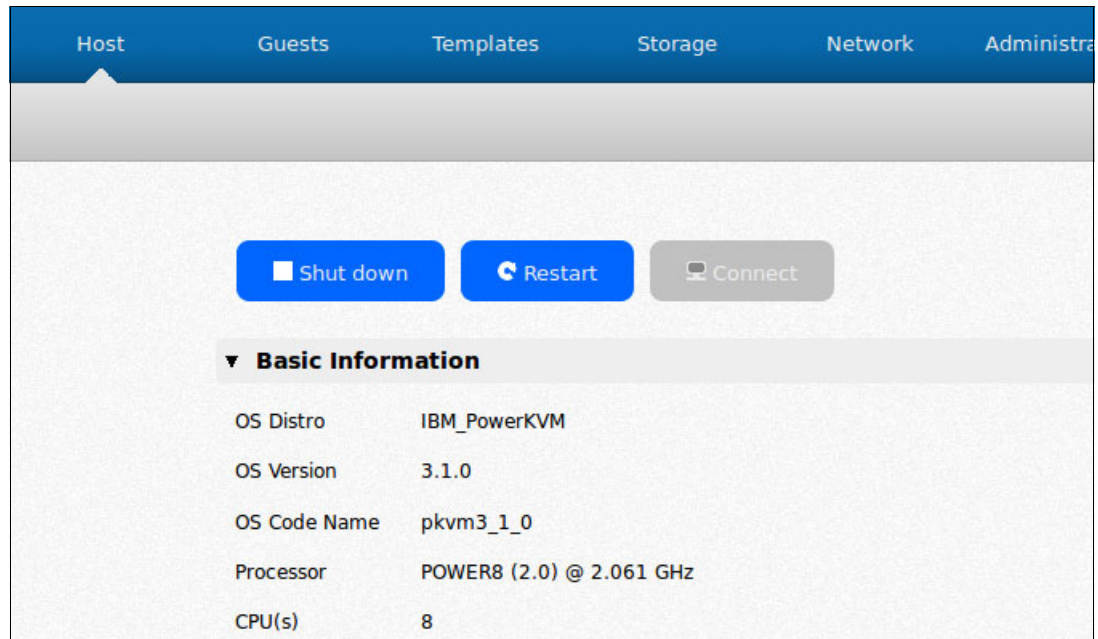


Figure 3-4 Host tab information

Figure 3-5 on page 65 shows the host system statistic in an easy-to-read graphic pane with the following fields:

- ▶ CPU
- ▶ Memory
- ▶ Disk I/O
- ▶ Network I/O



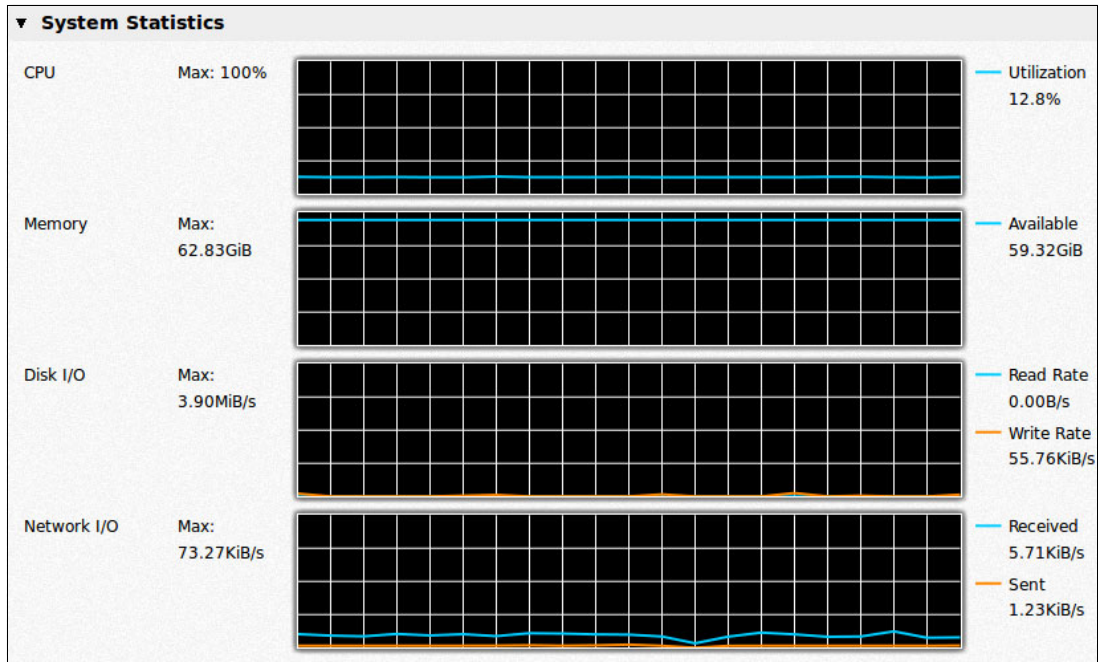


Figure 3-5 Host system statistics

Figure 3-6 shows the software update system and repository manager. By using the software update system, it is possible to know all available packages to update. Click **Update All** to update the whole system with a single click.

The repository manager allows any repository installed in the system to be enabled, disabled, or removed. It is also possible to add a new repository and to edit an existing one.

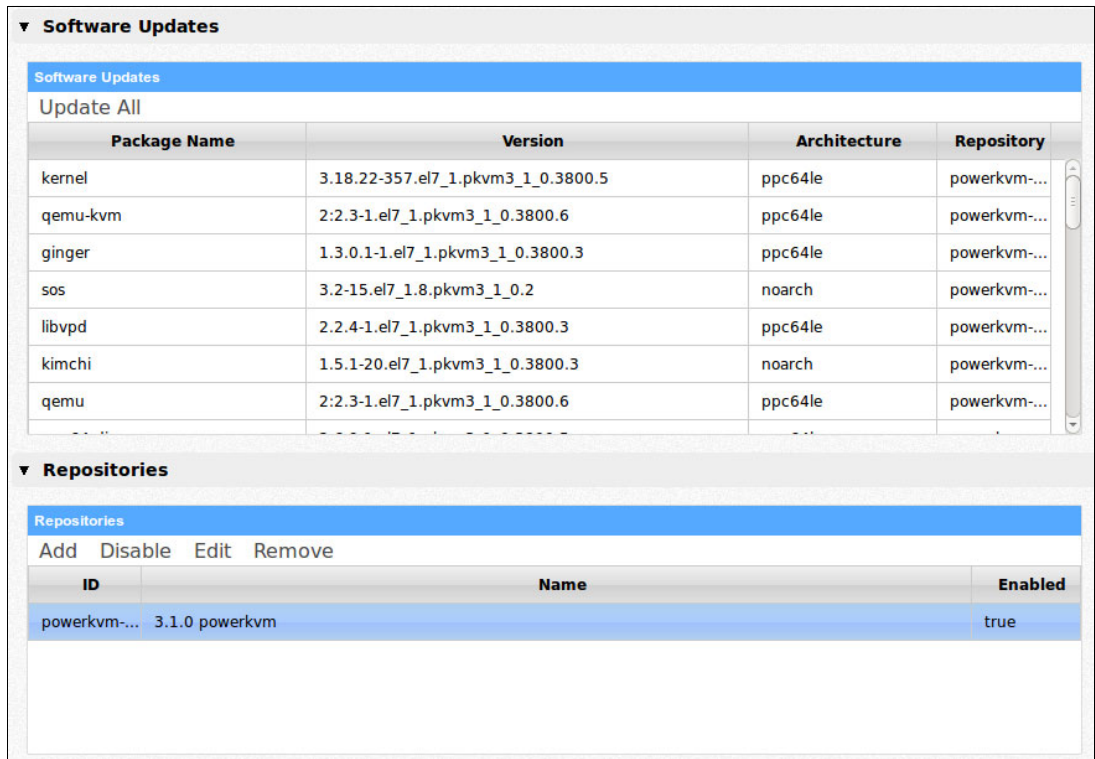


Figure 3-6 Package update and repository manager

Figure 3-7 shows how to add a new repository.

Based on Figure 3-6 on page 65, click **Add**, type a unique identifier in the Identifier field, complete the Name field, type the repository path in the URL field, select **Repository is a mirror** if necessary, then, click **Add**.

Kimchi checks whether the given repository is valid before adding it. Note that yum variables can be used. Kimchi knows how to expand them.

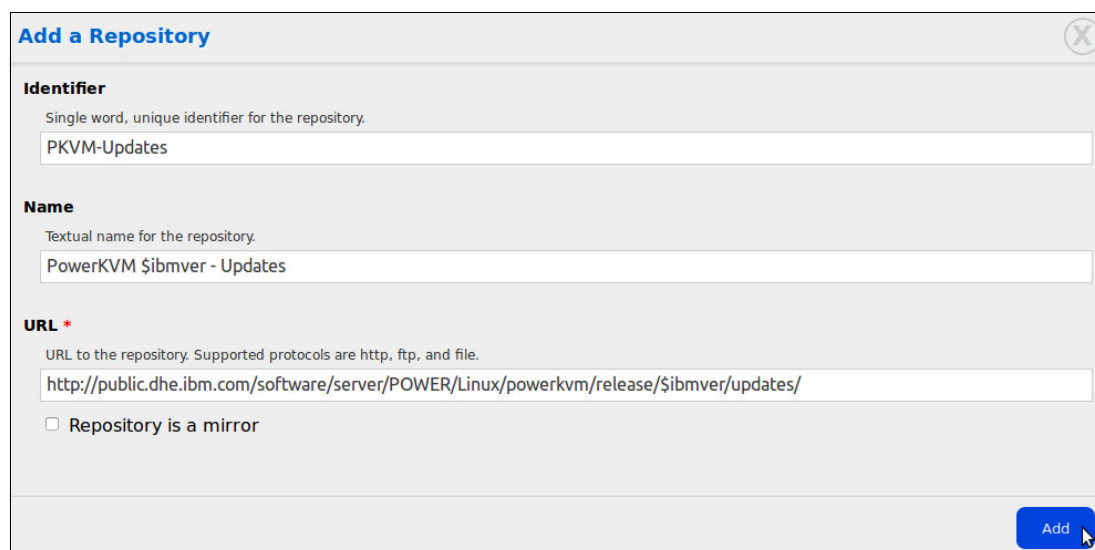


Figure 3-7 Adding a new repository

Figure 3-8 shows how to generate an SOS debug report by using Kimchi Debug Reports. Click **Generate** to open the debug report dialog box.

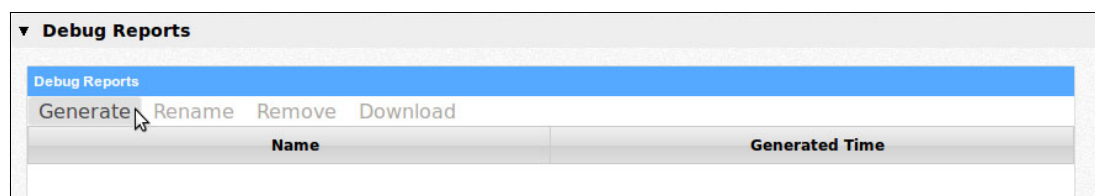
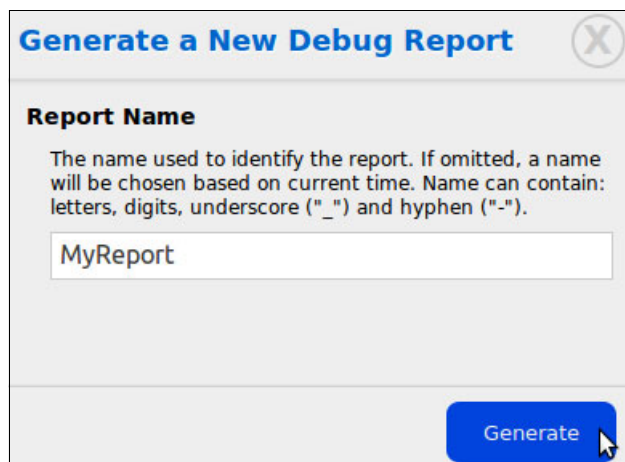


Figure 3-8 Generating a new SOS Report

Figure 3-9 shows the Generate a New Debug Report dialog box. Type the name in the Report Name field and click **Generate**.



**Generate a New Debug Report** [X]

**Report Name**

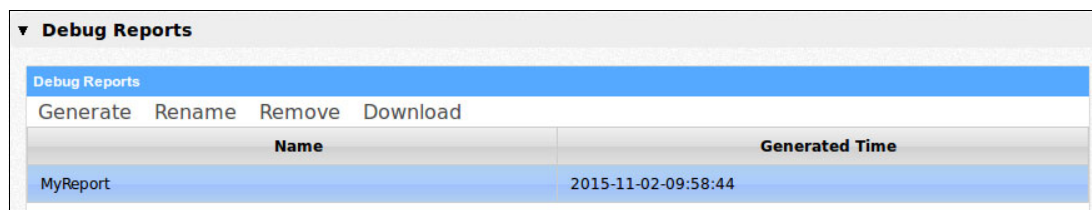
The name used to identify the report. If omitted, a name will be chosen based on current time. Name can contain: letters, digits, underscore (" \_") and hyphen (" -").

MyReport

Generate

Figure 3-9 Generating a new SOS Report

Figure 3-10 shows all debug reports listed in Kimchi. It is possible to rename, remove, and download the report.





▼ Debug Reports	
Debug Reports	
Generate	Rename Remove Download
Name	Generated Time
MyReport	2015-11-02-09:58:44

Figure 3-10 List of reports created

## 3.3 Storage pool

A storage pool is necessary to allocate space for guests' disks. Storage pools support the following media:

- ▶ Fibre Channel
- ▶ Internet Small Computer System Interface (iSCSI)
- ▶ Network File System (NFS)
- ▶ Logical volume
- ▶ Local directory

Host	Guests	Templates	Storage	Network	Administration		
Name	%Used	State	Location	Type	Capacity	Allocated	Actions
ISO	3%		/var/lib/kimchi/Isos	dir	9.7G	320.7M	Actions ▾
default	1%		/var/lib/libvirt/images	dir	878.9G	4.5G	Actions ▾

Define a New Storage Pool

1. Storage Pool Name

The name used to identify the storage pools, and it should not be empty.

stor1

2. Storage Pool Type

DIR

3. Storage Path

The path of the Storage Pool. Each Storage Pool must have a unique path.  
Kimchi will try to create the directory when it does not already exist in your system.

/srv/images

Create

Figure 3-12 Local directory storage pool

Figure 3-13 shows how to activate a newly created storage pool by using the Actions menu. Then, click **Activate**.

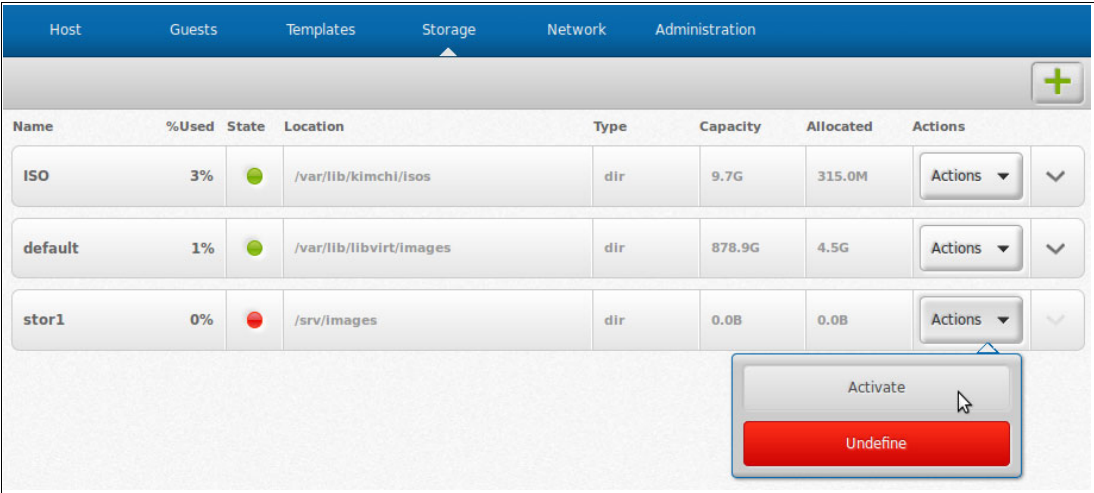
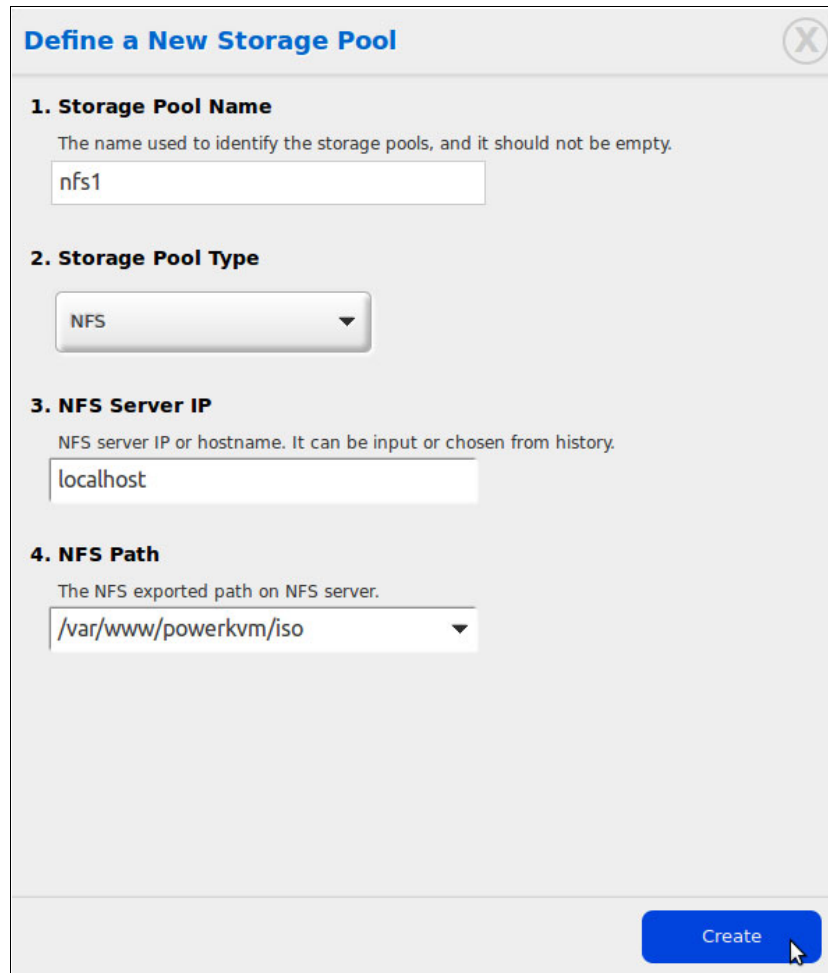


Figure 3-13 Activating a new storage pool

### 3.3.2 NFS

Figure 3-14 shows how to create a file-based storage pool by using NFS, which can be shared between multiple servers.

Type the name of the storage pool in the **Storage Pool Name** field, select **NFS** as the Storage Pool Type, enter the NFS Server IP field, enter the remote directory in the NFS Path field, and click **Create**.



The screenshot shows a web-based configuration window titled "Define a New Storage Pool" with a close button (X) in the top right corner. The window contains four numbered sections:

- 1. Storage Pool Name**: A text input field containing "nfs1". Above the field is the instruction: "The name used to identify the storage pools, and it should not be empty."
- 2. Storage Pool Type**: A dropdown menu with "NFS" selected.
- 3. NFS Server IP**: A text input field containing "localhost". Above the field is the instruction: "NFS server IP or hostname. It can be input or chosen from history."
- 4. NFS Path**: A dropdown menu with "/var/www/powerkvm/iso" selected. Above the field is the instruction: "The NFS exported path on NFS server."

At the bottom right of the window is a blue button labeled "Create" with a mouse cursor hovering over it.

Figure 3-14 NFS storage pool

### 3.3.3 iSCSI

Figure 3-15 shows how to create an iSCSI storage pool. New volumes in that pool need to be created by using your storage server management interface. It is not possible to do that from Kimchi.

To create the iSCSI storage pool, enter the Storage Pool Name field, select **iSCSI** as the Storage Pool Type, enter the IP address in the iSCSI Server field, enter the iSCSI target in the Target field, and click **Create**.

To use authentication, select **Add iSCSI Authentication**, and type the user name and password on the respective fields before clicking **Create**.

**Define a New Storage Pool**

**1. Storage Pool Name**  
The name used to identify the storage pools, and it should not be empty.  
iscsi1

**2. Storage Pool Type**  
iSCSI

**3. iSCSI Server**  
iSCSI server IP or hostname. It should not be empty.  
9.40.193.34 Port

**4. Target**  
The iSCSI target on iSCSI server  
iqn.1986-03.com.ibm.2145:myiscsi

☐ Add iSCSI Authentication

Create

Figure 3-15 iSCSI storage pool



iSCSI volumes are detected from an iSCSI storage pool, as shown in Figure 3-16.



Figure 3-16 iSCSI volumes list

Figure 3-17 shows a template that uses an iSCSI volume. In contrast to other storage pools, when an iSCSI and Fibre Channel storage pool are used, the guest is created by using a template that points to that specific volume.

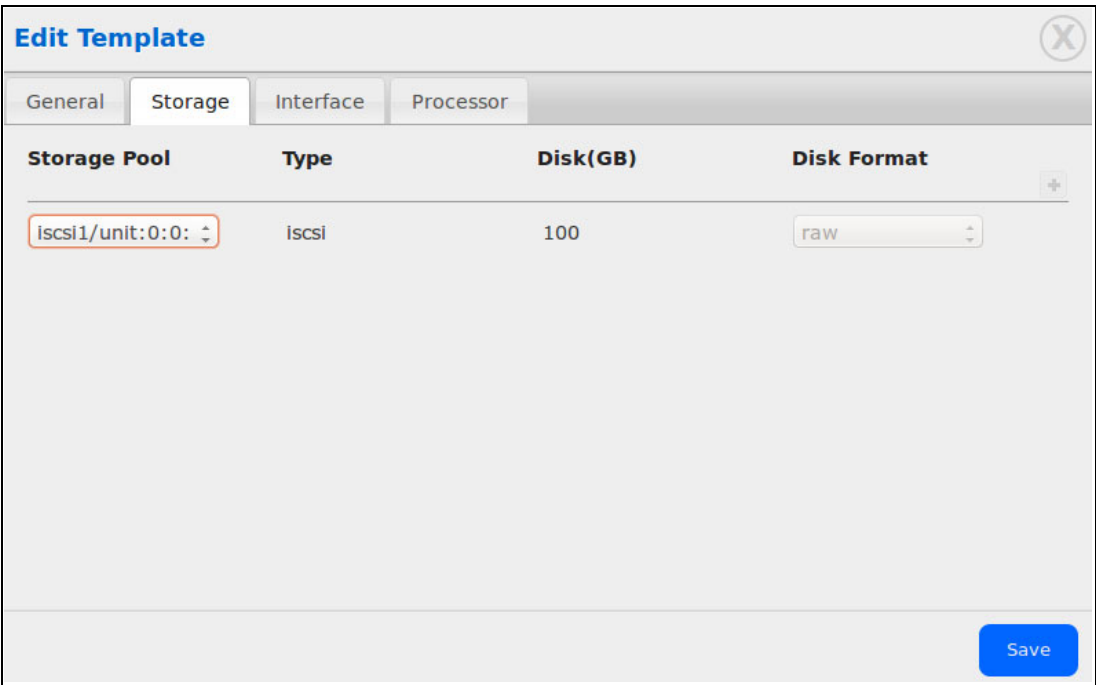
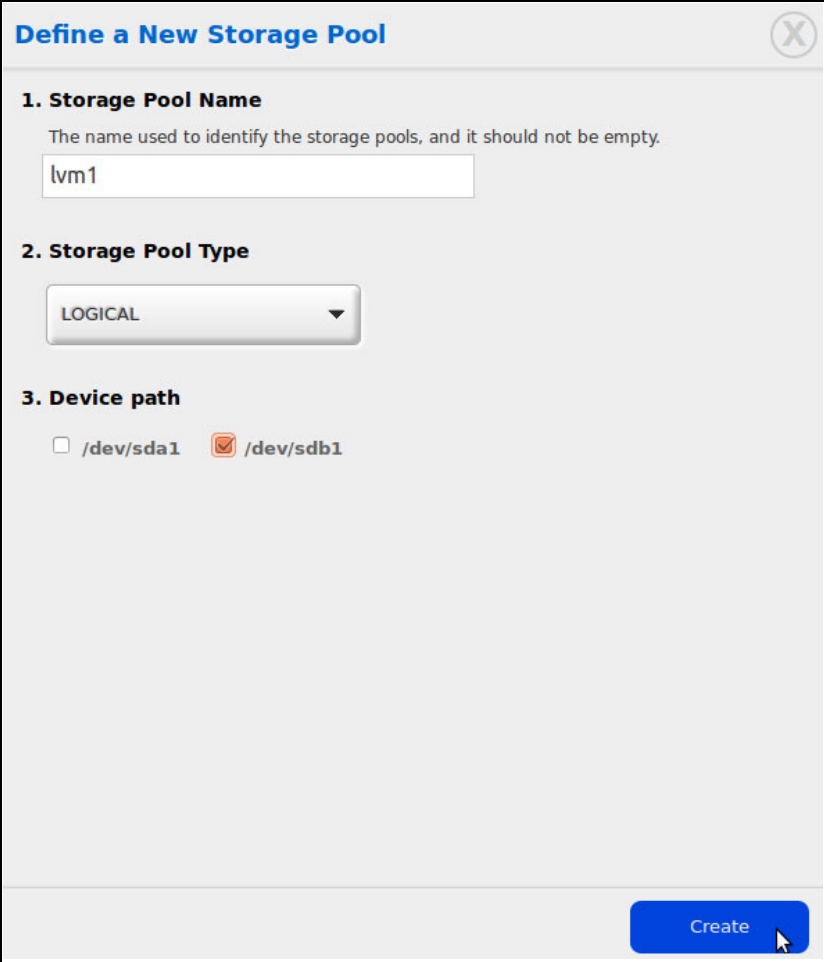


Figure 3-17 Creating a template using an iSCSI volume

### 3.3.4 Logical volume

Figure 3-18 shows a Logical Volume Manager (LVM) storage pool. It creates a volume group by using existing physical devices, and logical volumes are created for the guests by using that volume group.

To create an LVM storage pool, enter the name in the Storage Pool Name field, select **LOGICAL** as the Storage Pool Type, select the physical devices for the volume group, and then click **Create**.



**Define a New Storage Pool**

**1. Storage Pool Name**  
The name used to identify the storage pools, and it should not be empty.  
lvm1

**2. Storage Pool Type**  
LOGICAL

**3. Device path**  
☐ /dev/sda1 ☒ /dev/sdb1

Create

Figure 3-18 LVM storage pool

Figure 3-19 shows the volume group listed in the Storage tab. By clicking **Actions**, then clicking **Extend**, it is possible to add more devices to the volume group.



Figure 3-19 LVM storage pool

### 3.4 Network

In Kimchi, you can create a NAT network, a bridged network, or an isolated network. Section 6.2, “Network virtualization” on page 164, describes the differences between NAT and bridge networks.

Figure 3-20 shows the Network tab in Kimchi, where a default NAT network can be found. New networks can be created by clicking **Add**.

**Note:** Network cannot be stopped if a guest is running on that.

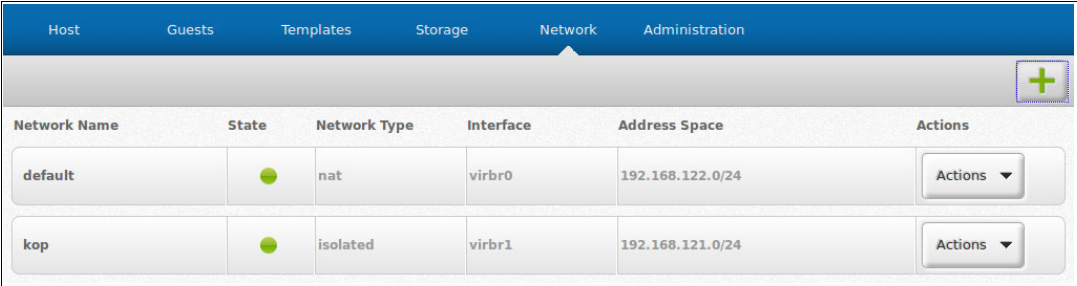


Figure 3-20 Network tab

Figure 3-21 demonstrates how to start or activate a network. Guests using a stopped network can be run, but they will not have network access. New networks must be started before using them.

**Note:** A new guest cannot be created by using a template configured with a network that is stopped. Make sure that the network is started when creating and running guests.

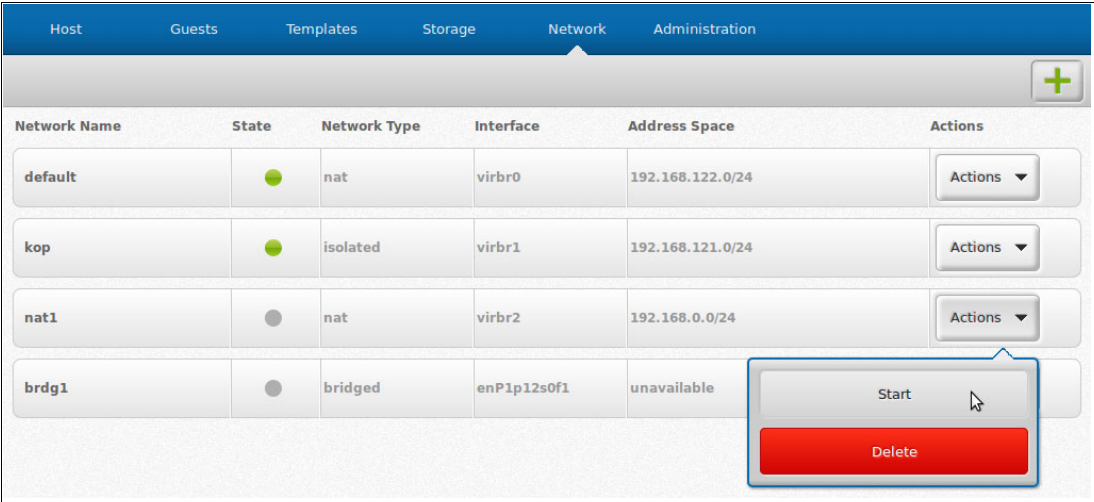
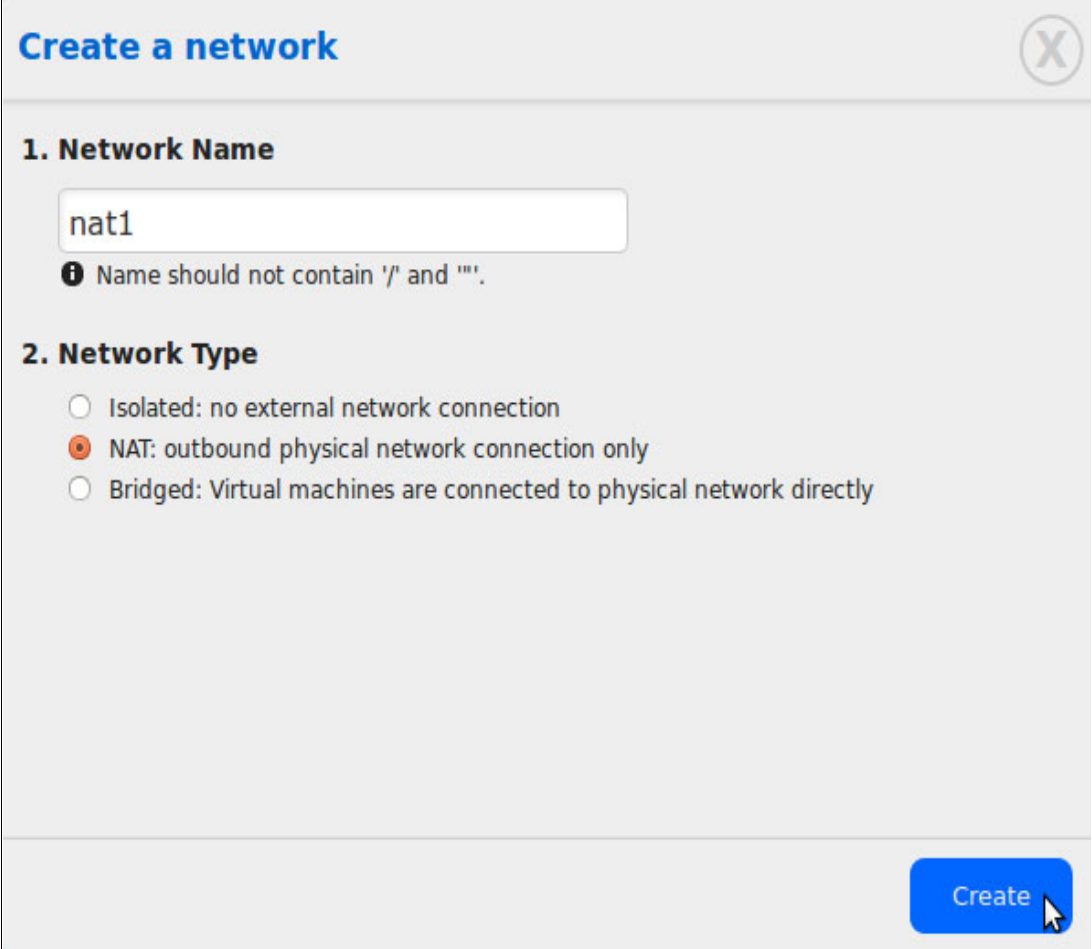


Figure 3-21 Starting a network

### 3.4.1 NAT network

Figure 3-22 shows how to create a new NAT network. Enter the network name, click **NAT**, and then click **Create**. That creates a new IPv4 NAT network that uses a private address range of 256 addresses.



The screenshot shows a web interface titled "Create a network" with a close button (X) in the top right corner. The interface is divided into two main sections:

- 1. Network Name**: A text input field contains the name "nat1". Below the field is an information icon (i) and a note: "Name should not contain '/' and '\"'".
- 2. Network Type**: Three radio button options are listed:
  - ☐ Isolated: no external network connection
  - ☒ NAT: outbound physical network connection only
  - ☐ Bridged: Virtual machines are connected to physical network directly

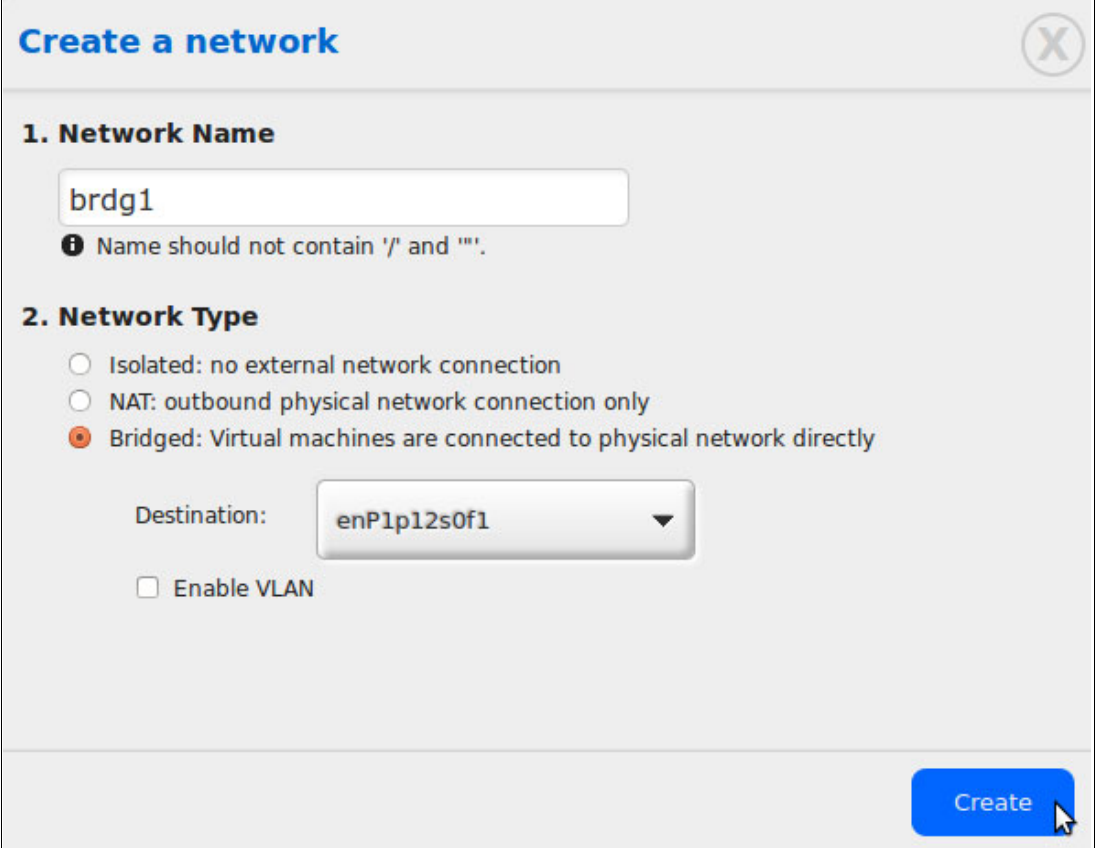
A blue "Create" button with a mouse cursor is located at the bottom right of the dialog.

Figure 3-22 Creating a new NAT network

### 3.4.2 Bridged network

Figure 3-23 shows how to create a bridged network. Type the network name, select **Bridged**, select a network interface from the **Destination** list, and then click **Create**. To enable VLAN support, before clicking **Create**, select the **Enable VLAN** check box, and type the VLAN ID in the VLAN ID field.

**Note:** The network interface used as the destination of a bridge must be configured and set up on the host.



The screenshot shows a 'Create a network' dialog box with a close button (X) in the top right corner. The dialog is divided into two main sections:

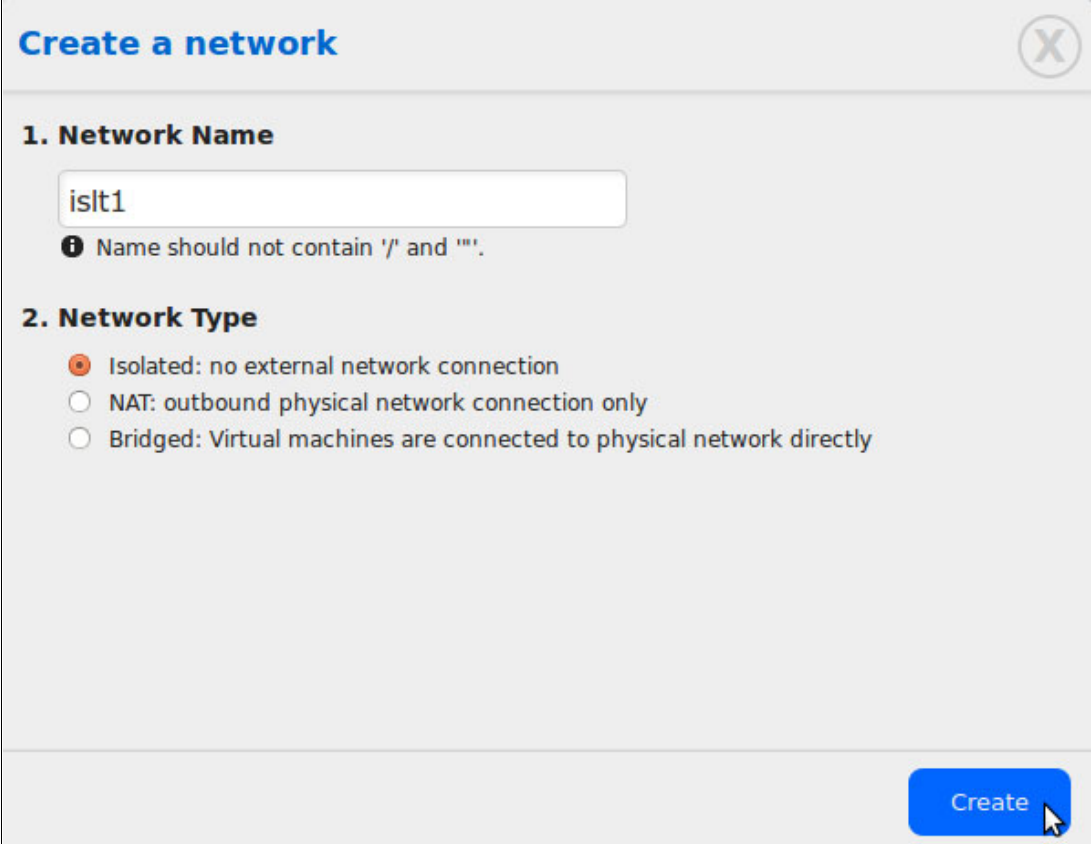
- 1. Network Name**: A text input field contains 'brdg1'. Below it, a message icon (i) states: 'Name should not contain '/' and ''.'.
- 2. Network Type**: Three radio buttons are present:
  - ☐ Isolated: no external network connection
  - ☐ NAT: outbound physical network connection only
  - ☒ Bridged: Virtual machines are connected to physical network directly

Below the network type options, there is a 'Destination:' label followed by a dropdown menu showing 'enP1p12s0f1'. Below the dropdown is a checkbox labeled 'Enable VLAN' which is currently unchecked. At the bottom right of the dialog is a blue 'Create' button with a mouse cursor hovering over it.

Figure 3-23 Creating a bridged network

### 3.4.3 Isolated network

Figure 3-24 shows how to create an isolated network. Guests using that network cannot access or be accessed from external systems.



**Create a network**

**1. Network Name**

islt1

**2. Network Type**

- ☒ Isolated: no external network connection
- ☐ NAT: outbound physical network connection only
- ☐ Bridged: Virtual machines are connected to physical network directly

Create

Figure 3-24 Creating an isolated network

## 3.5 Templates

In Kimchi, a *template* is a set of basic parameters necessary to create new guests. It is designed to store configuration details that multiple guests, having that configuration in common, can be created efficiently. Some of the parameters contained in a template are listed:

- ▶ Path to the operating system (local or remote)
- ▶ Number of CPUs
- ▶ CPU topology
- ▶ Memory size
- ▶ Disk size
- ▶ Storage pool to be used
- ▶ Networks to be used

Every virtual machine created with Kimchi is based on a template.

When created, templates can be edited or removed, but it is important to notice that the relationship between a template and a guest does not exist after the virtual machine was created. It means that changes to existing templates have no effect to existing guests created from those templates.

### 3.5.1 Create a new template

Figure 3-25 shows the Templates tab. Click **Add** to create a new template and choose between local or remote ISOs.

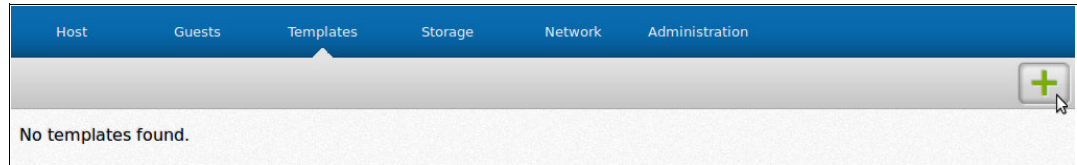


Figure 3-25 Template tab

Figure 3-26 shows the media source dialog box. Click **Local ISO Image**.

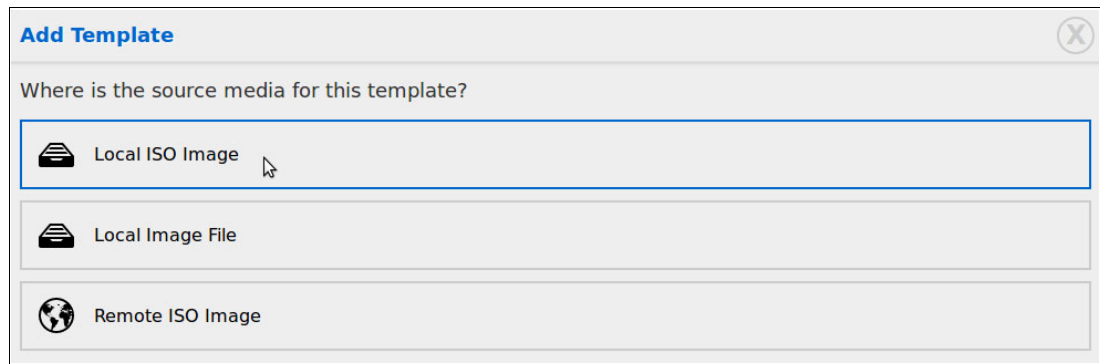


Figure 3-26 Selecting the Local ISO Image

Figure 3-27 shows the ISOs that are available. Pick the operating system and click **Create Templates from Selected ISO**.

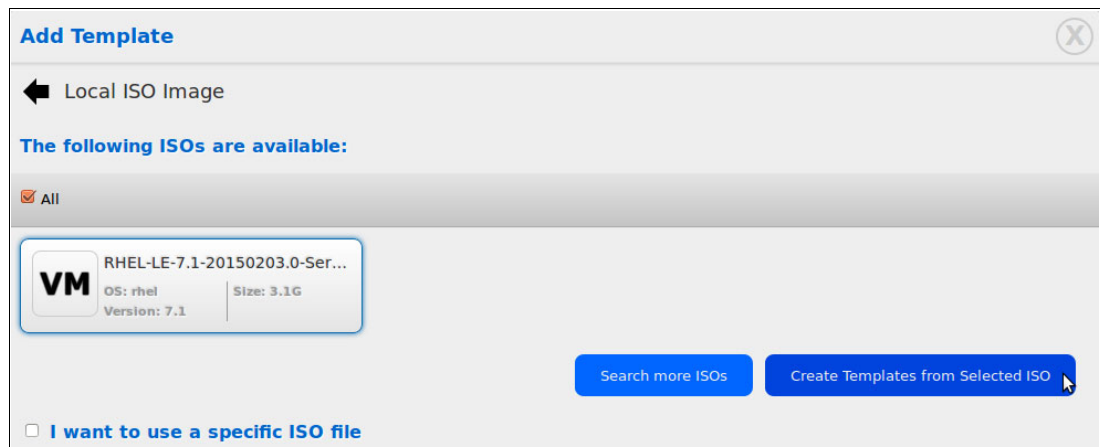


Figure 3-27 Selecting the operating system



### 3.5.2 Edit an existing template

It is necessary to edit a template to modify any parameter. Figure 3-28 shows a template and its Actions menu. Click **Edit** to change the template configuration.

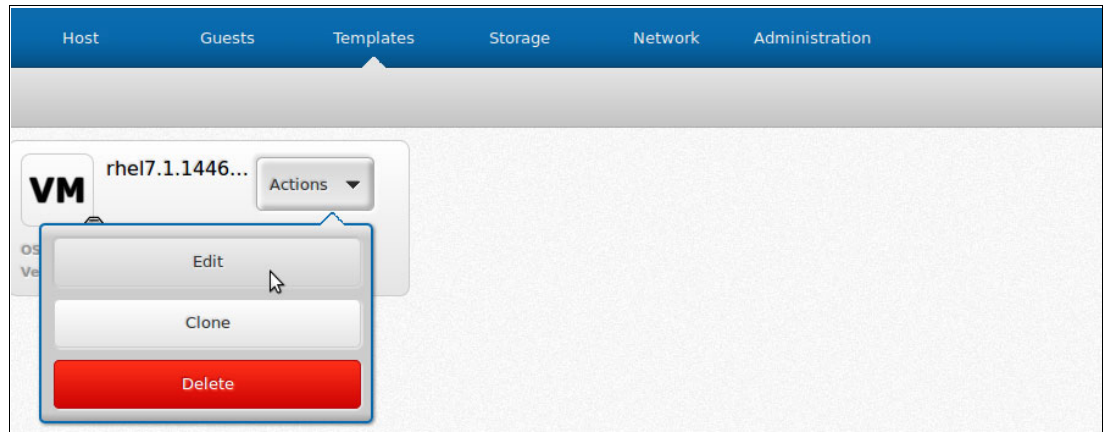
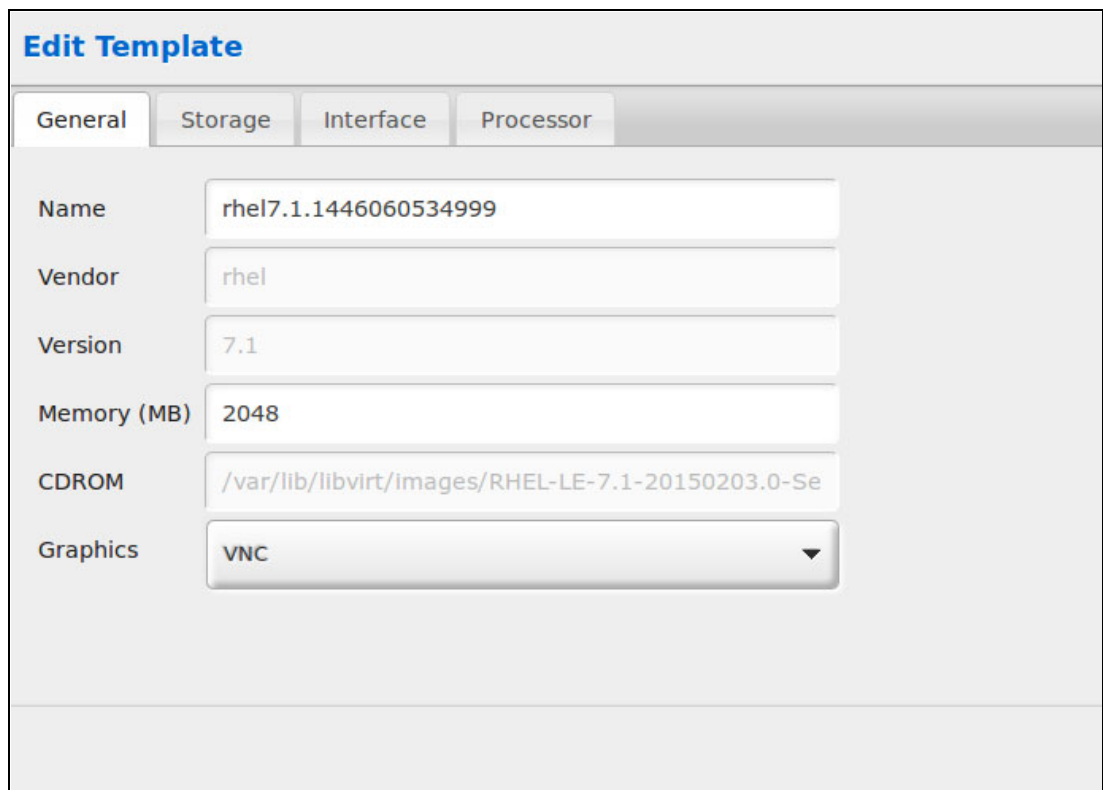


Figure 3-28 Editing a template

Figure 3-29 shows the Edit Template dialog box. In the General tab, it is possible to change the template name, the amount of memory, and the graphics (currently only VNC is supported).

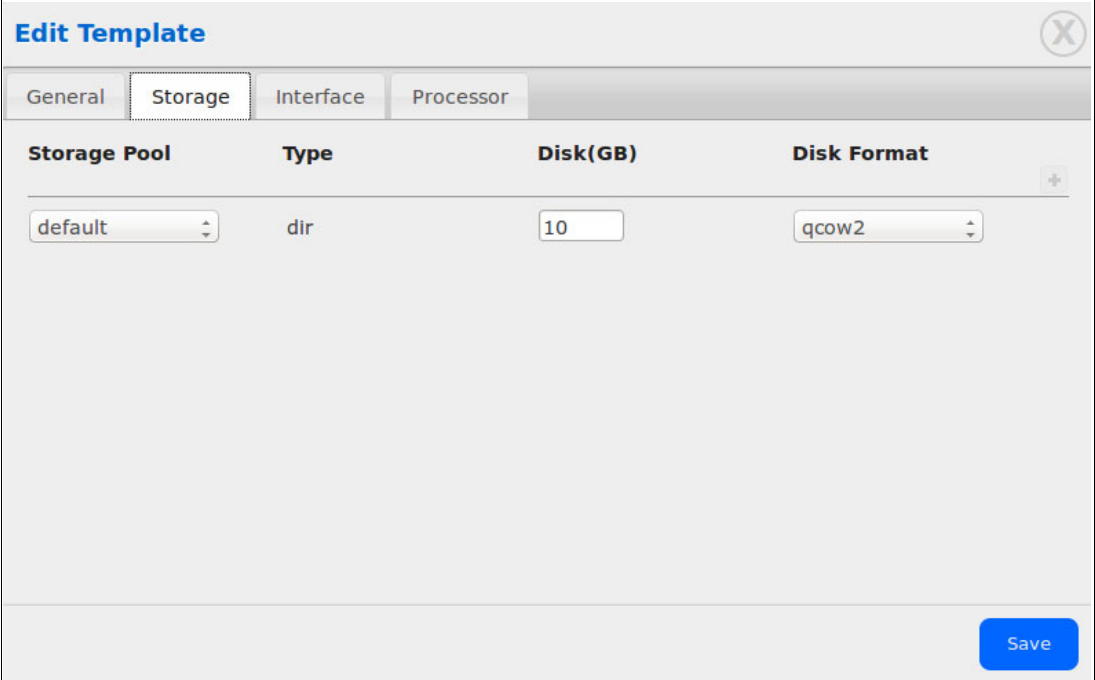
A screenshot of the 'Edit Template' dialog box. The 'General' tab is selected. The form contains the following fields:

- Name: rhel7.1.1446060534999
- Vendor: rhel
- Version: 7.1
- Memory (MB): 2048
- CDROM: /var/lib/libvirt/images/RHEL-LE-7.1-20150203.0-Se
- Graphics: VNC (dropdown menu)

Figure 3-29 Template editor

## Storage tab

Figure 3-30 shows the Storage tab. It allows users to specify how much space will be allocated from the selected storage pool and the disk format required. The storage pools listed here are those created in 3.3, “Storage pool” on page 67.



**Edit Template**

General **Storage** Interface Processor

Storage Pool	Type	Disk(GB)	Disk Format
default	dir	10	qcow2

Save

Figure 3-30 Storage tab in the template editor

## Interface tab

Figure 3-31 shows the Interface tab, where the guest network can be created or removed. The networks listed here are those created in 3.4, “Network” on page 75.

Network	Type
default	network

Figure 3-31 Network tab in the template editor

## Processor tab

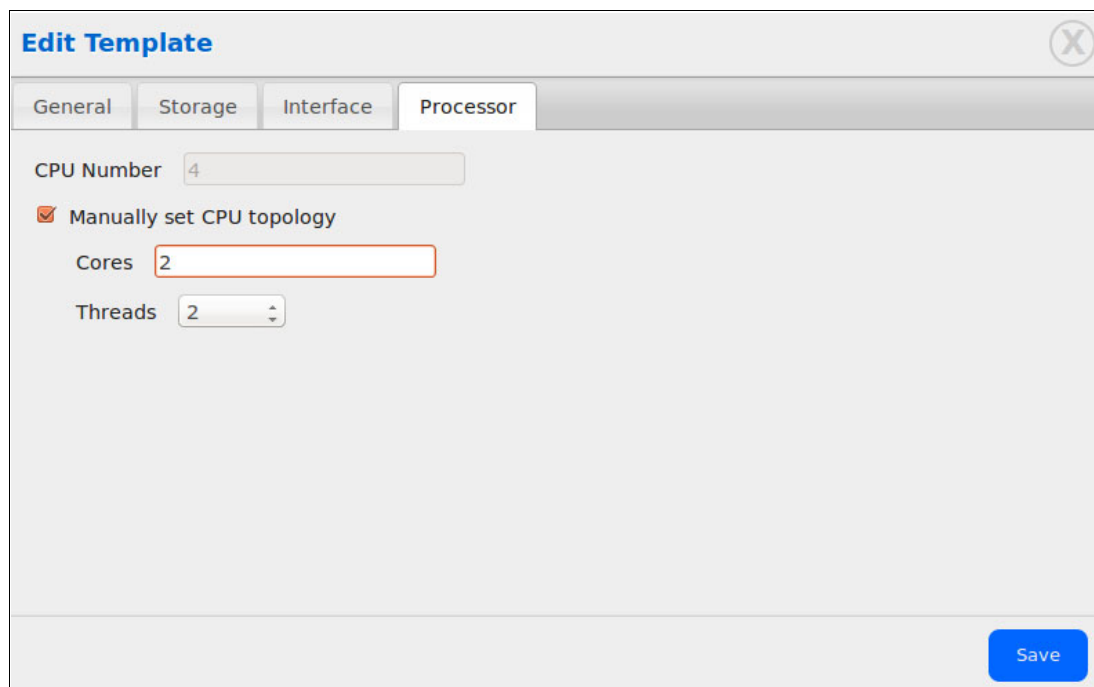
Figure 3-32 shows the Processor tab. The number of virtual CPUs can be set directly in the CPU Number field.

CPU Number

☐ Manually set CPU topology

Figure 3-32 Processor tab in template editor

Another option is to set the CPU topology as shown in Figure 3-33. In this particular case, the processor has two cores with two threads each.



The screenshot shows the 'Edit Template' window with the 'Processor' tab selected. The 'CPU Number' is 4. The 'Manually set CPU topology' checkbox is checked. The 'Cores' field is set to 2, and the 'Threads' dropdown is set to 2. A 'Save' button is at the bottom right.

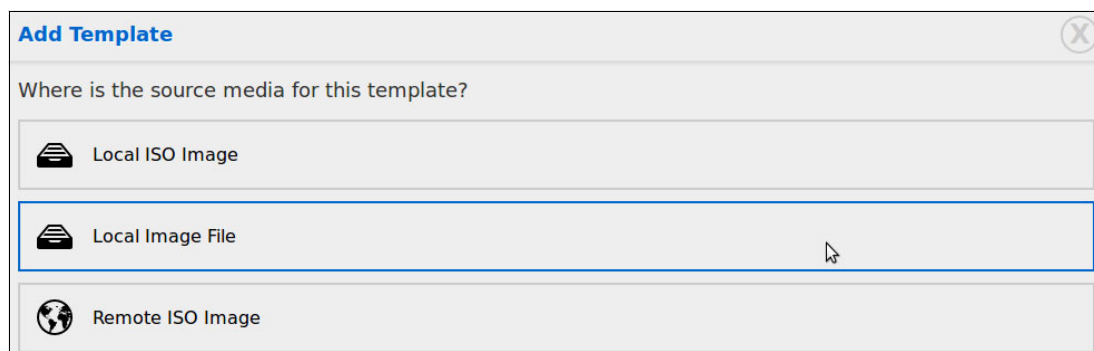
Figure 3-33 Editing CPU topology in template editor

After changes are made, click **Save** to persist the template.

### 3.5.3 Create a template from an existing disk image

Based on a disk image, it is possible to create a template from it. This feature is particularly useful when needed to reuse a disk image from another server.

Based on Figure 3-25 on page 80, click **Add** for a new template and select **Local Image File**, as shown in Figure 3-34.



The screenshot shows the 'Add Template' window. The question 'Where is the source media for this template?' is at the top. Three options are listed: 'Local ISO Image', 'Local Image File' (which is selected and highlighted with a blue border), and 'Remote ISO Image'.

Figure 3-34 Selecting the local image file

Figure 3-35 shows how to add the template. Type the path to the image that is going to be used as source in the File Path field, then click **Create**.

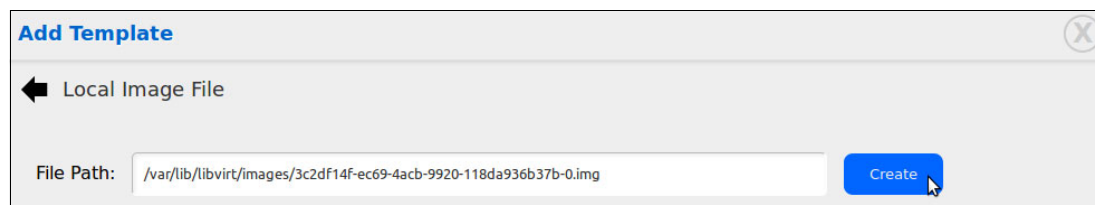


Figure 3-35 Image disk file path

When a new template is created, new guests created based on that template will start at the same state where the original disk image was when the template was created.

## 3.6 Guests

Managing guests involve creating or editing guests from a template, and then starting or stopping them as necessary.

### 3.6.1 Create a new guest

To create a new guest, create a template as described in 3.5.1, “Create a new template” on page 80.

On the Guests tab, click **add** as shown in Figure 3-36.

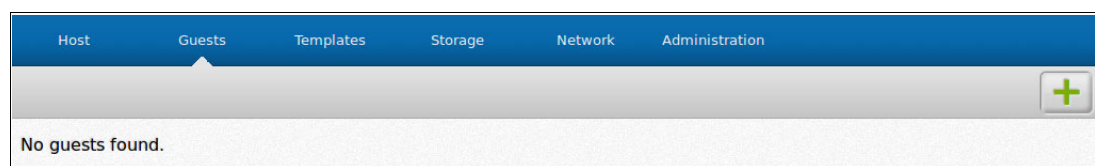
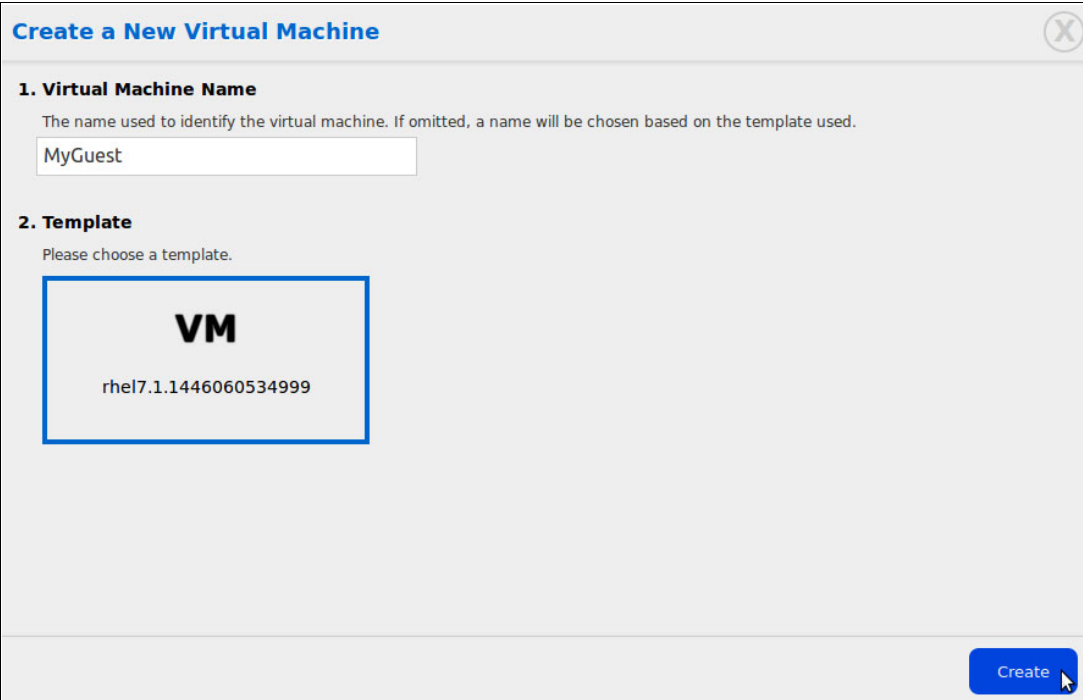


Figure 3-36 The guest tab

Figure 3-37 shows the dialog box to create a new guest. Enter a name into the Virtual Machine Name field, select the **Template**, as seen on 3.5, “Templates” on page 79, then click **Create**.



**Create a New Virtual Machine**

**1. Virtual Machine Name**  
The name used to identify the virtual machine. If omitted, a name will be chosen based on the template used.  
MyGuest

**2. Template**  
Please choose a template.  
VM  
rhel7.1.1446060534999

Create

Figure 3-37 Creating a new virtual machine

## 3.6.2 Guest management

All guests are listed in the Guest tab. Each row displays a guest state summary. This makes it easier to understand how guests are performing. The fields displayed are:

- ▶ Guest name
- ▶ CPU
- ▶ Disk I/O
- ▶ Network I/O
- ▶ Livetile
- ▶ Action box to control the virtual machine

In the action box, users can control the guest. The actions that are available depend on the current guest state. When stopped, users can access a group of functions according to that state. To start the guest system, click **Start** or select **Actions**, then select **Start**, as shown in Figure 3-38 on page 87.

It is still possible to edit the guest, delete it completely, or clone a particular guest.

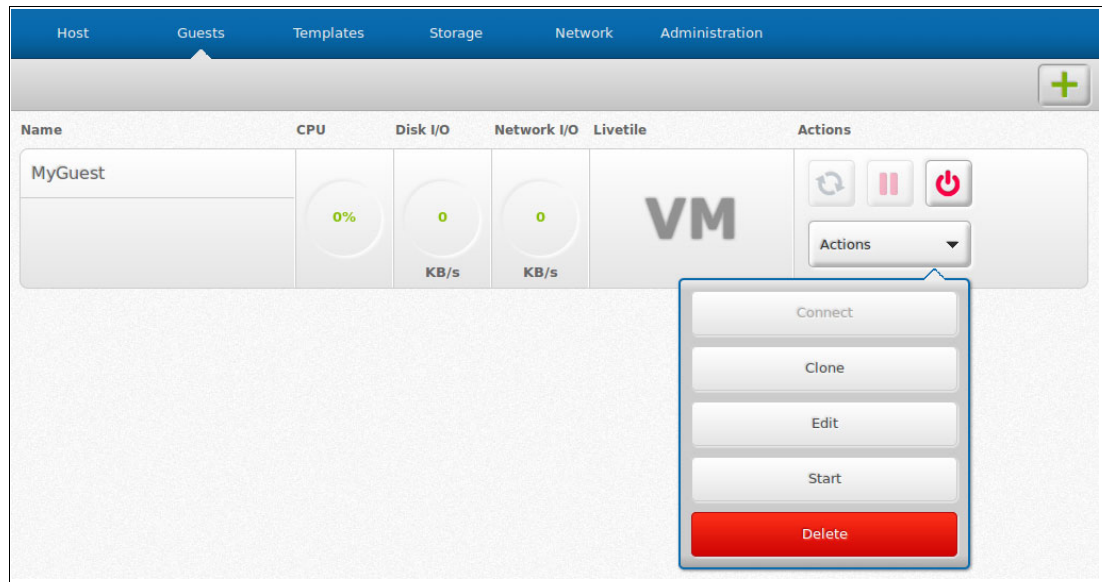


Figure 3-38 Guest control buttons for stopped guest

### Edit a stopped guest

Figure 3-39 shows how to edit a stopped guest. By selecting **Actions** then clicking **Edit** as shown in Figure 3-38, a new dialog box is displayed. The first tab in the Edit window is General, where users can change the guest name, the amount of memory, and the number of virtual CPUs for that guest.

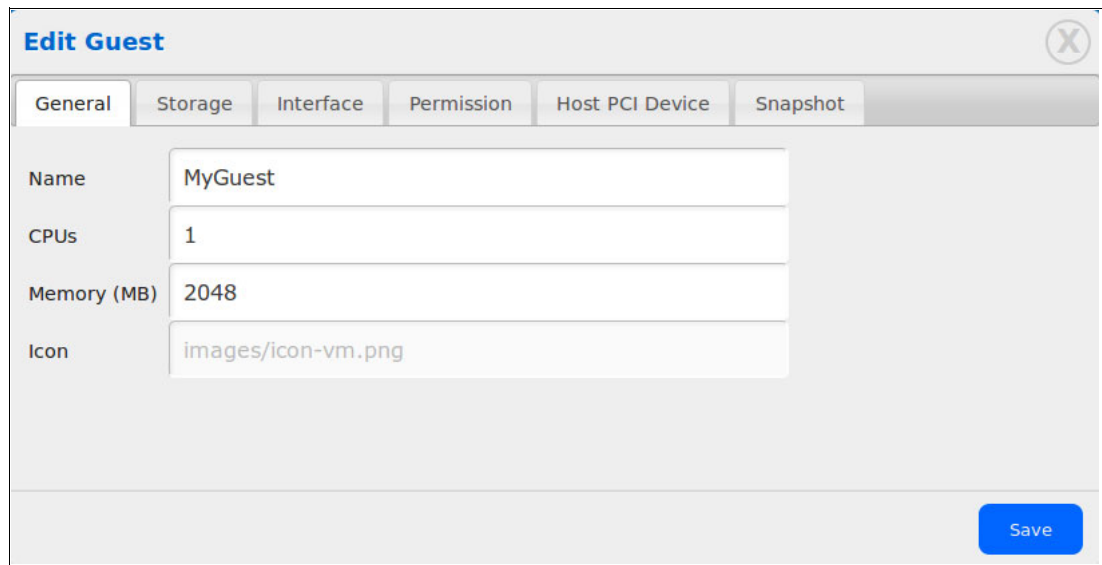


Figure 3-39 Edit a stopped guest

### Storage tab

Figure 3-40 show the Storage tab. It is possible to attach or detach storage to guests such as existing disks and CD-ROM.

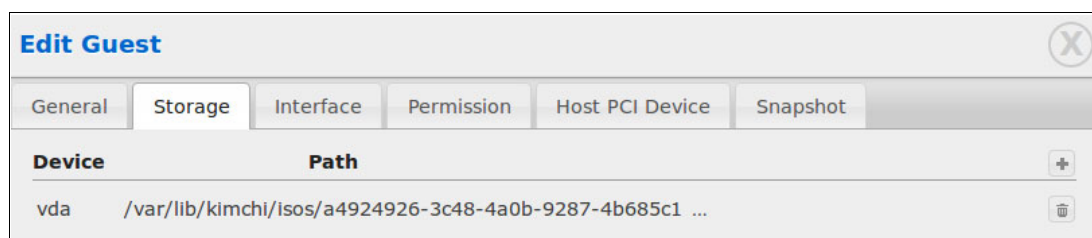


Figure 3-40 Storage tab interface

Figure 3-41 shows how to attach an existing disk image to a particular guest.

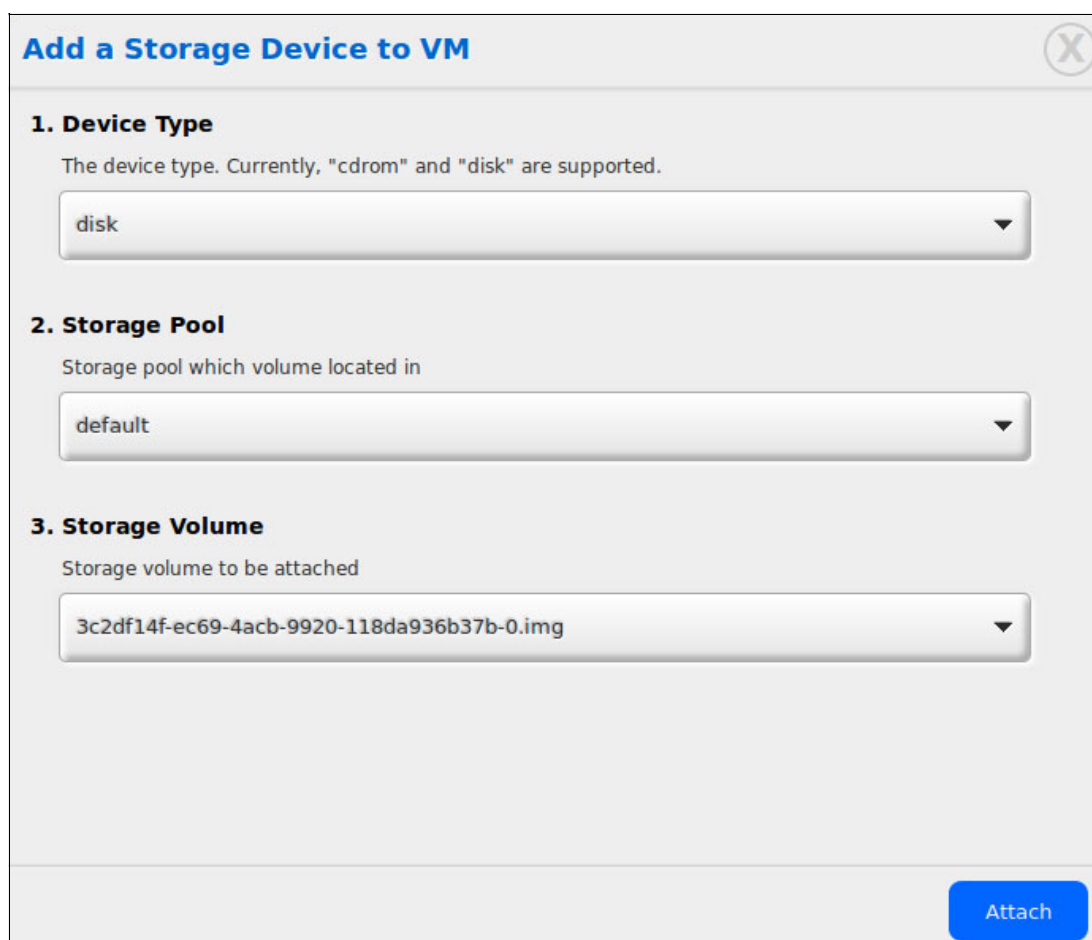


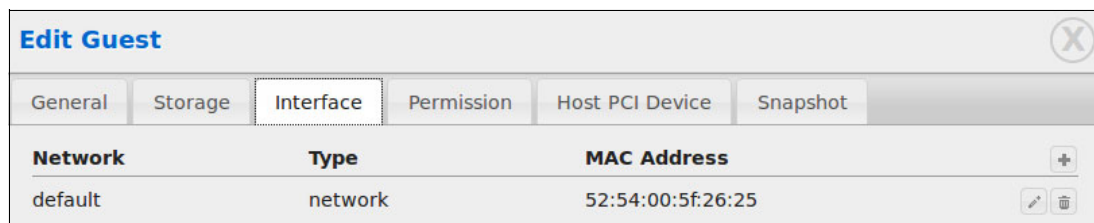
Figure 3-41 Attach an existing volume



### Interface tab

Figure 3-42 shows the Interface tab. This allows users to create, edit, or remove networks. To create a new network interface, select the network created in 3.4, “Network” on page 75, enter the MAC address in the MAC Address field, then click **Save**.

**Note:** MAC address will be chosen automatically if the MAC address field is left blank.

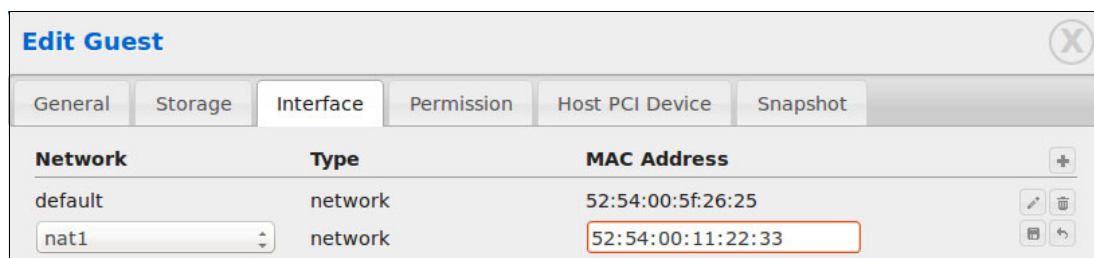


Network	Type	MAC Address
default	network	52:54:00:5f:26:25

Figure 3-42 Interface tab

Figure 3-43 shows how to create a new interface. To persist the changes, click **Save** at the right of the MAC address field. It is possible to undo the changes by clicking **Undo**.

Click the **pen** icon to change the MAC address of any interface. To delete a network interface, click the **trash can**.



Network	Type	MAC Address
default	network	52:54:00:5f:26:25
nat1	network	52:54:00:11:22:33

Figure 3-43 Adding a network

### Permission tab

Figure 3-44 on page 90 shows the Permission tab, where the Kimchi administrator grants permissions to users or groups of a particular guest. Such control allows users to access their guest only. Select the user or the group from **Available system users and groups** and click the **right arrow**. To undo the process, select the users from **Selected system users and groups**, and click the **left arrow**.

**Note:** Refer to “User Management tool” on page 102 to know more about user roles in Kimchi.

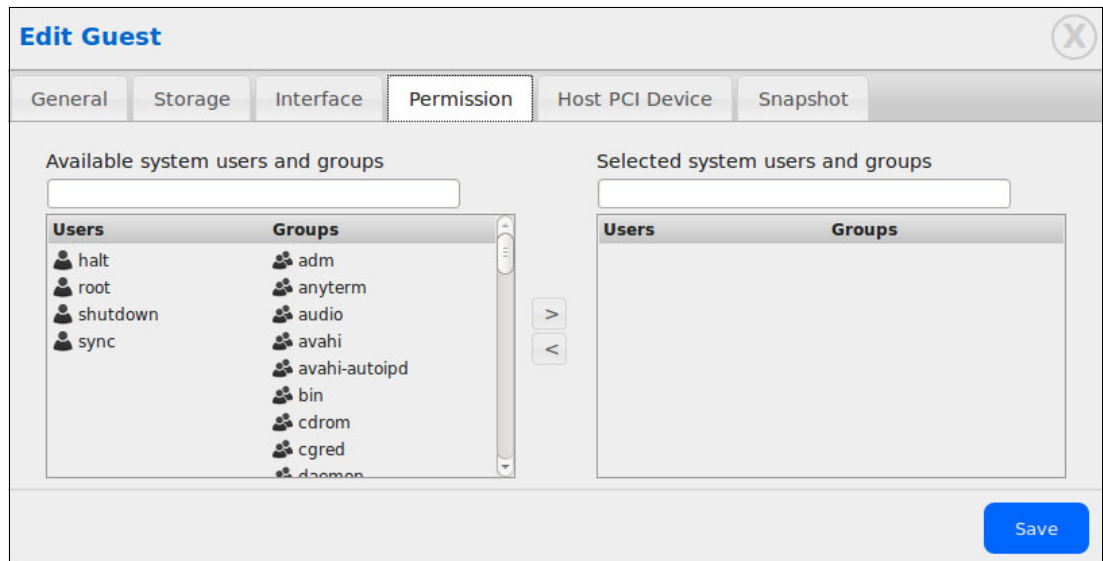


Figure 3-44 Permission tab

### Host PCI Device tab

Figure 3-45 shows the Host PCI Device tab for PCI pass-through. Kimchi supports simple devices as well as multifunction devices pass-through. Click **Add** to attach a device to a guest. That device will be automatically detached from host when guest is started.

**Note:** The Host PCI Device tab does not list devices that are currently attached to another guest. The device must be detached before becoming available.

To detach a device from the guest, click **Remove**.

**Note:** If a multifunction device is selected, all of its functions are automatically attached to or detached from the guest.

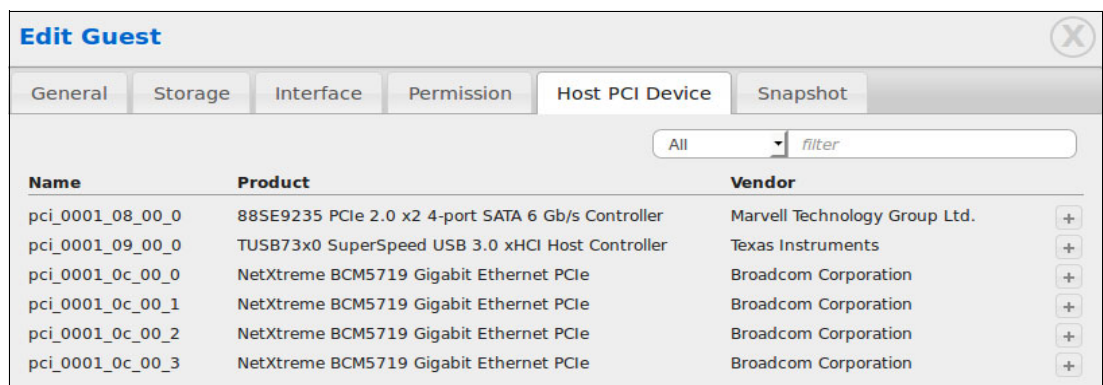


Figure 3-45 Host PCI device tab

### Snapshot tab

Figure 3-46 on page 91 shows the Snapshot tab in Kimchi. Snapshot is an important feature that stores the disk state in the moment the snapshot is taken. Multiple snapshots can be taken by clicking **Add**.

When needed click **Revert**, left to the **trash can**, to apply the snapshot.

**Note:** Reverting a guest to a particular snapshot loses any data modified after the snapshot creation.



Figure 3-46 Snapshot tab

## Start and stop a guest

The first time a new guest is started, it tries to load the operating system (OS) from the disk. Because there is no OS installed on the disk, it will fail, so it will load from the ISO. After that, it's possible to proceed with the OS installation.

Click the Livetile box or select Actions and click Connect to open the guest window in a new browser tab to install your OS.

## Edit a running guest

Figure 3-47 shows the Guest tab with a running guest. The menu displays more options and the actions buttons are now enabled to reset, pause, or shut down the guest system. Clone is disabled because it is not possible to clone a running guest.

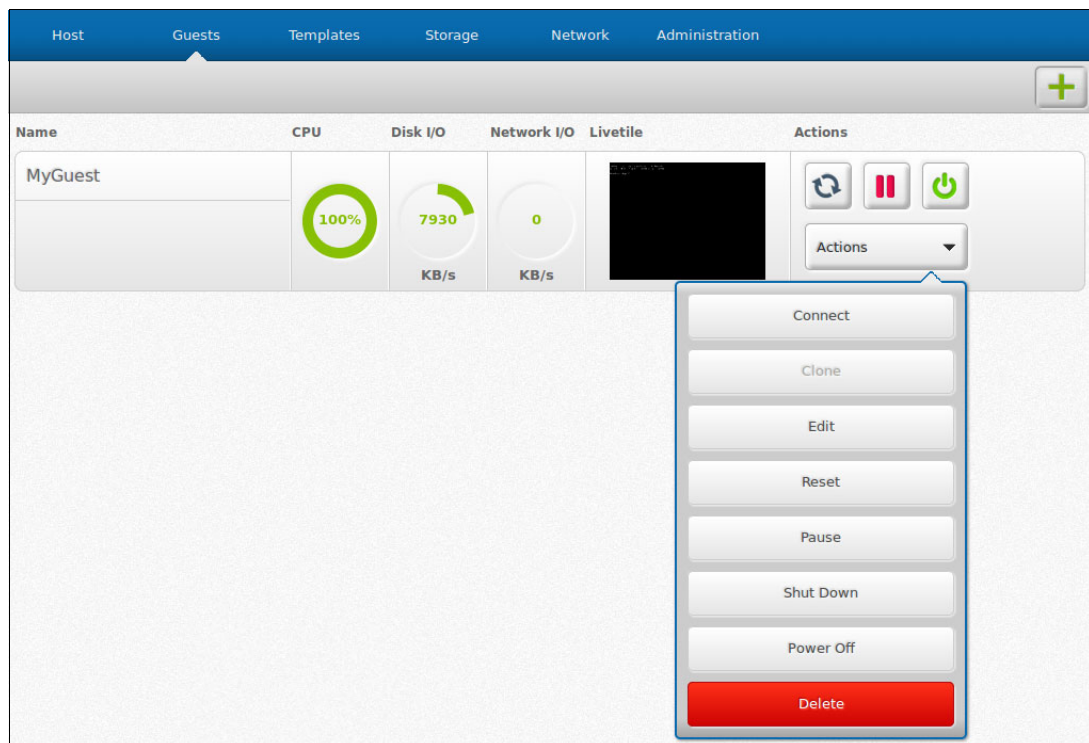


Figure 3-47 Guest control buttons for a running guest

Based on Figure 3-47 on page 91, click **Edit**. The General tab is read-only when the guest is running, so go to the Interface tab.

### Storage hotplug

It is possible to attach a disk image in a running guest. Example 3-1 displays all disks attached in a particular guest.

#### Example 3-1 Guest disks

---

```
[root@localhost ~]# ls /dev/[sv]d*  
/dev/sda /dev/sda1 /dev/vda /dev/vda1 /dev/vda2 /dev/vda3
```

---

Figure 3-48 shows how to add a storage device. Based on Figure 3-47 on page 91, click **Edit**, then select the Storage tab, select the **Device Type**, then select the storage pool where the image is allocated, select the image, and click **Attach**.

**Add a Storage Device to VM**

**1. Device Type**  
The device type. Currently, "cdrom" and "disk" are supported.  
disk

**2. Storage Pool**  
Storage pool which volume located in  
default

**3. Storage Volume**  
Storage volume to be attached  
3c2df14f-ec69-4acb-9920-118da936b37b-0.img

Attach

Figure 3-48 Attach a storage to a running guest

Example 3-2 lists all disks in a guest, including the attached disks.

#### Example 3-2 Guest disks

---

```
[root@localhost ~]# ls /dev/[sv]d*  
/dev/sda /dev/sda1 /dev/vda /dev/vda1 /dev/vda2 /dev/vda3 /dev/vdb /dev/vdb1  
/dev/vdb2 /dev/vdb3
```

---

## Network interface hotplug

Kimchi supports network interface hotplug. Example 3-3 shows all existing interfaces from a particular guest before hotplugging the new interface.

Example 3-3 Guest network interfaces

```
[root@localhost ~]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    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 qlen 1000
    link/ether 52:54:00:5f:26:25 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.4/24 brd 192.168.122.255 scope global dynamic eth0
        valid_lft 3285sec preferred_lft 3285sec
    inet6 fe80::5054:ff:fe5f:2625/64 scope link
        valid_lft forever preferred_lft forever
```

Figure 3-49 shows how to add a new network interface. Click **Add**, select the **Network**, type the MAC address in the MAC Address field or leave it blank for automatic fill, and click **Save**.

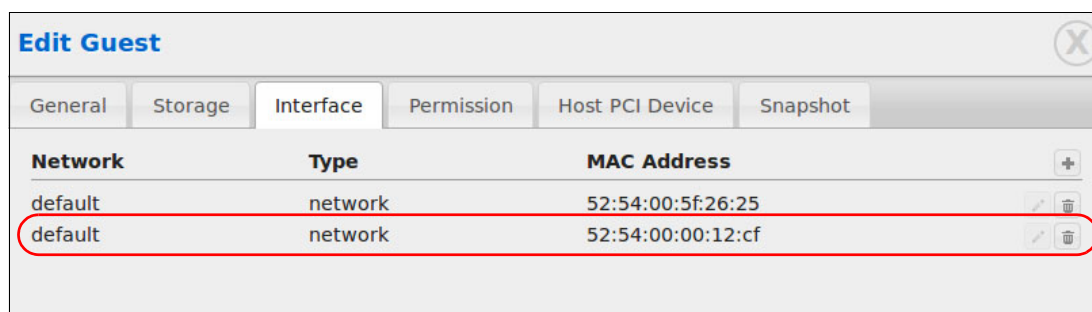


Figure 3-49 Adding a new network interface

Example 3-4 shows the new interface listed by the `ip addr` command.

Example 3-4 Guest network interfaces

```
[root@localhost ~]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    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 qlen 1000
    link/ether 52:54:00:5f:26:25 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.4/24 brd 192.168.122.255 scope global dynamic eth0
        valid_lft 3233sec preferred_lft 3233sec
    inet6 fe80::5054:ff:fe5f:2625/64 scope link
        valid_lft forever preferred_lft forever
4: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
```

```

link/ether 52:54:00:01:2e:e0 brd ff:ff:ff:ff:ff:ff
5: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
link/ether 52:54:00:00:12:cf brd ff:ff:ff:ff:ff:ff

```

## PCI hotplug

To hotplug a PCI device with Kimchi, open the Host PCI Device tab, click **Add**, and the device will be automatically detached from the host to be attached to the guest.

Example 3-5 shows all devices in the guest before hotplugging.

Example 3-5 Listing all devices in the guest

```

[root@localhost ~]# lspci
00:01.0 Ethernet controller: Red Hat, Inc Virtio network device
00:02.0 SCSI storage controller: Red Hat, Inc Virtio block device
00:03.0 Unclassified device [00ff]: Red Hat, Inc Virtio memory balloon
00:04.0 VGA compatible controller: Device 1234:1111 (rev 02)
00:0f.0 USB controller: NEC Corporation uPD720200 USB 3.0 Host Controller (rev 03)

```

Figure 3-50 shows how to attach a PCI device to the guest by simply clicking **Add**. After it is attached, the button becomes a minus sign.

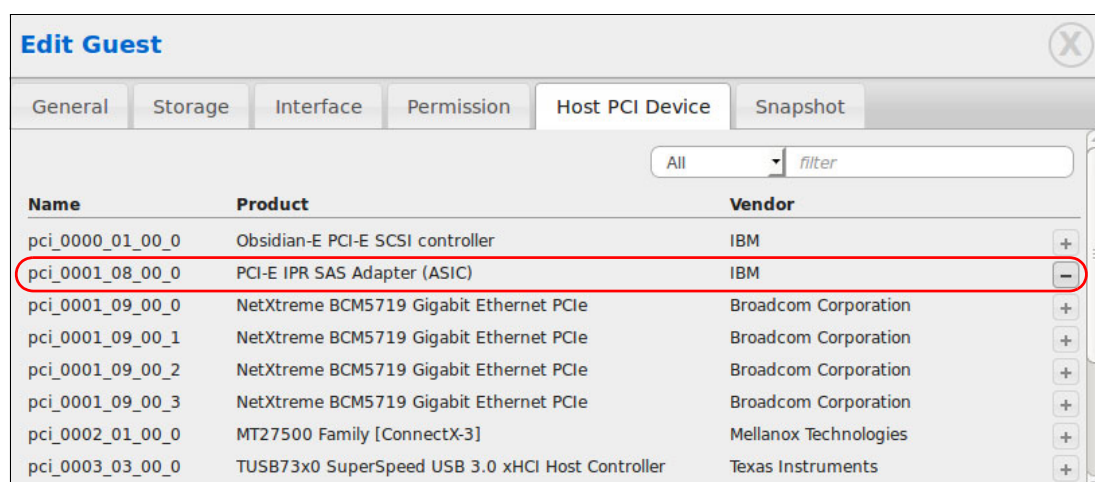


Figure 3-50 List of PCI devices

Example 3-6 shows the device PCI-E IPR SAS Adapter attached to the guest.

Example 3-6 Listing all devices in guest

```

[root@localhost ~]# lspci
00:01.0 Ethernet controller: Red Hat, Inc Virtio network device
00:02.0 SCSI storage controller: Red Hat, Inc Virtio block device
00:03.0 Unclassified device [00ff]: Red Hat, Inc Virtio memory balloon
00:04.0 VGA compatible controller: Device 1234:1111 (rev 02)
00:09.0 RAID bus controller: IBM PCI-E IPR SAS Adapter (ASIC) (rev 01)
00:0f.0 USB controller: NEC Corporation uPD720200 USB 3.0 Host Controller (rev 03)

```

Figure 3-51 shows how to unplug the device. Click **Remove** to detach the device from the guest. The button becomes a plus sign again.

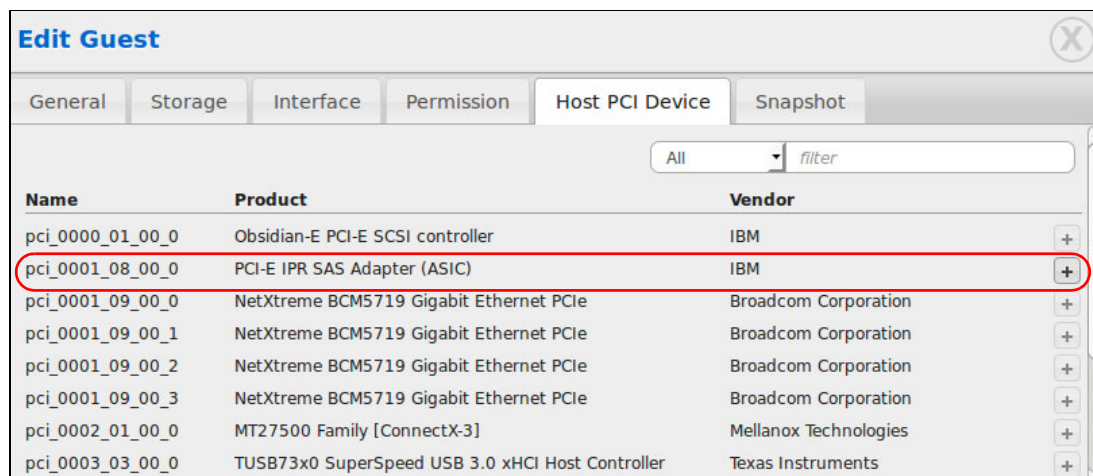


Figure 3-51 List of PCI devices

Example 3-7 shows the PCI list after detachment.

Example 3-7 Listing all devices in guest

```
[root@localhost ~]# lspci
00:01.0 Ethernet controller: Red Hat, Inc Virtio network device
00:02.0 SCSI storage controller: Red Hat, Inc Virtio block device
00:03.0 Unclassified device [00ff]: Red Hat, Inc Virtio memory balloon
00:04.0 VGA compatible controller: Device 1234:1111 (rev 02)
00:0f.0 USB controller: NEC Corporation uPD720200 USB 3.0 Host Controller (rev 03)
```

## 3.7 Accessing a graphical interface

PowerKVM supports VNC to access a graphical interface console on the guest. Kimchi uses noVNC, which is a web-based VNC client.

### 3.7.1 noVNC

noVNC can be accessed either by clicking the preview image on the Guests tab or by clicking **Actions** and then clicking **Connect**.

That opens a new browser tab or window that shows the graphical interface, which enables the user to interact with the guest machine.

**Note:** Your browser needs to allow pop-up windows to get a noVNC window. If your browser blocks pop-up windows, you will not be able to see the noVNC window.

### 3.7.2 VNC

You can get a graphical display of a guest by using VNC. Each guest uses a different TCP port, 5900+N, where N is the display number.



Example 3-8 shows how to obtain the VNC display number for a guest named MyGuest, using the **virsh** command on a root shell on the server.

*Example 3-8 Obtaining the VNC display number*

---

```
# virsh vncdisplay MyGuest
127.0.0.1:1
```

---

**Note:** Remember to open the VNC ports on the server firewall. The author suggests to set a password for the VNC access.

### 3.7.3 Custom keyboard layouts in Kimchi noVNC

When accessing a guest using a noVNC session from Kimchi, some non-US keyboards might not work. This is a known issue that was not fixed at the time of writing. As a workaround, a VNC server can be started inside the guest and accessed from any VNC client directly.

The XML guest files also allow to add a keymap attribute, as shown in Example 3-9 for a German keyboard, but this might not work for all keys.

*Example 3-9 keymap setting in guest XML*

---

```
<graphics type='vnc' port='-1' autoport='yes' listen='127.0.0.1' keymap='de'>
  <listen type='address' address='127.0.0.1'/>
</graphics>
```

---

Supported key maps can be found under `/usr/share/qemu/keymaps` in the PowerKVM host.

**Note:** Accessing the guest using the virsh console or a network connection is not an issue regarding non-US keyboards.

## 3.8 Ginger

*Ginger* is a host management plug-in for Kimchi, thus sharing the same user experience. In PowerKVM, Ginger is installed by default and can be accessed by clicking the Administration tab in Kimchi. All Ginger administration tools are listed below:

- ▶ Firmware Update
- ▶ Configuration Backup
- ▶ Network Configuration
- ▶ Power Options
- ▶ SAN Adapters
- ▶ Sensor Monitor
- ▶ SEP Configuration
- ▶ User Management



Figure 3-52 shows the Administration tab.

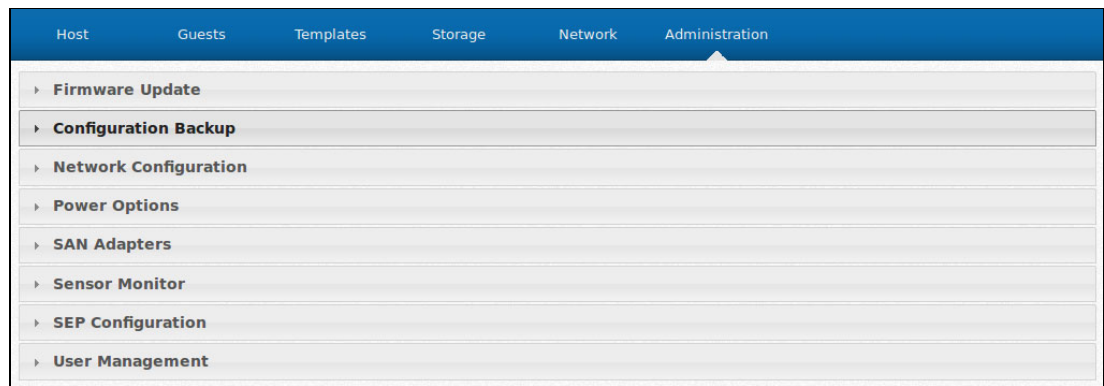


Figure 3-52 Administration tab

## Firmware Update

Figure 3-53 shows how to update the firmware using Ginger. On the Administration tab, click **Firmware Update**. Enter the path to the firmware in the Package Path field, and click **Update**.

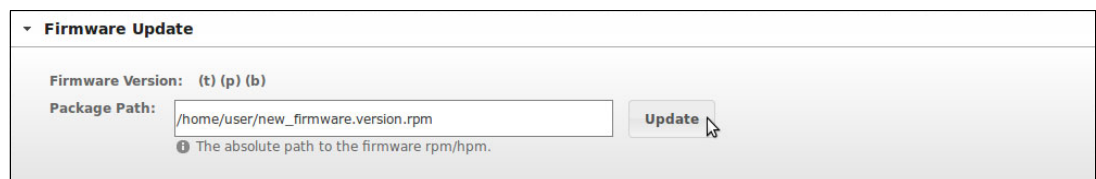


Figure 3-53 Firmware update tool

**Note:** Ginger does not show detailed information if any error event happens. For a more verbose firmware update, use the `update_flash` or `ipmitool` command directly.

To do a firmware update manually by using either the `update_flash` or `ipmitool` commands, go directly to IBM instructions in the Fix Central web page.

<https://www.ibm.com/support/fixcentral>

Figure 3-54 shows an example of how to select a firmware image from the IBM Fix Central website. Select **Power** in the Product Group drop-down menu, then select **Scale-out LC** in the Product drop-down menu, and it displays specific fields regarding your selection. Select your machine and click **Continue**.

Find product

Select product

Select the product below.

When using the keyboard to navigate the page, use the **Alt** and **down arrow** keys to navigate the selection lists.

Product Group\*

Power

Product\*

Scale-out LC

Select from Scale-out LC\*

8335-GCA

Select from 8335-GCA\*

All

Continue

Figure 3-54 Selecting an image from IBM Fix Central

### Configuration Backup

This tool is designed to back up the host configuration files.

Figure 3-55 shows the configuration backup tool. Click **Generate Default Backup** to have the following directories backed up automatically:

- ▶ /etc
- ▶ /var/spool/cron

**Note:** Generate Default Backup excludes /etc/init.d, /etc/rc.d, and /etc/rcN.d from the backup package.

Configuration Backup

Generate Default Backup

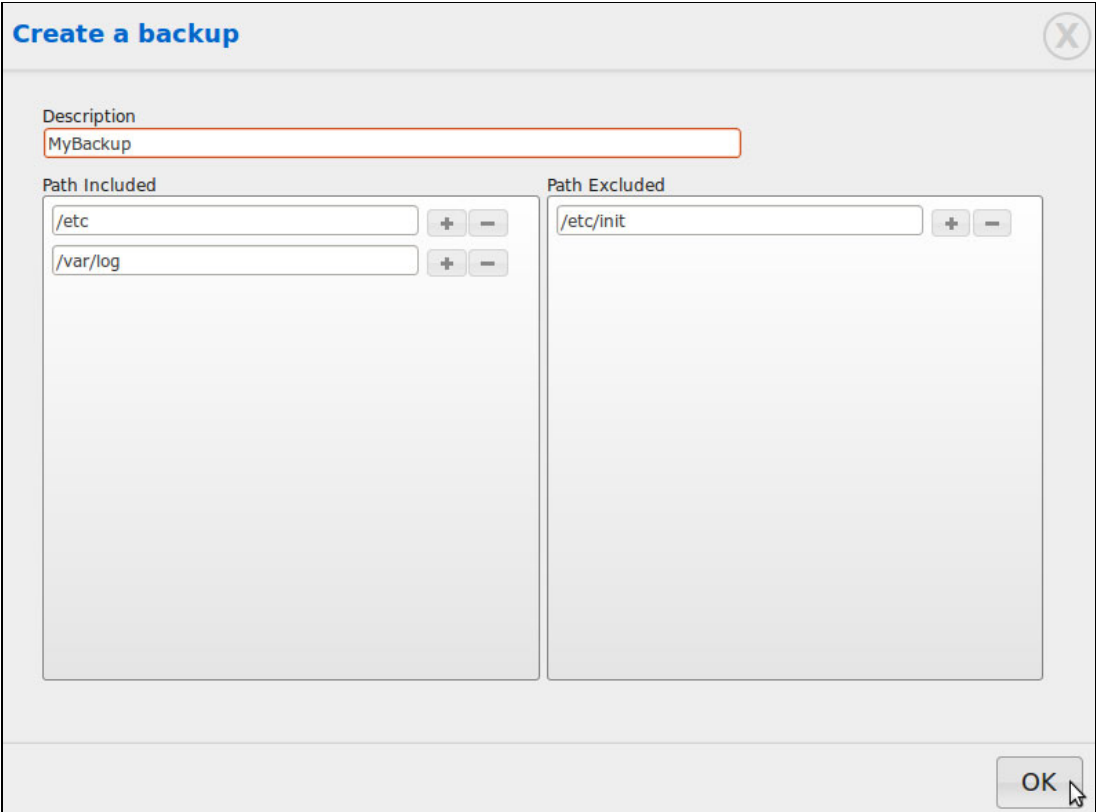
New Custom Backup

Batch Delete

File	Timestamp	
/var/lib/kimchi/ginger_backups/cdb14186-3e0c-4f1d-b2a9-ad1b894185c3.tar.gz	30/10/2015 11:03:31	<div>⌵ ✕</div>

Figure 3-55 Ginger configuration backup tool

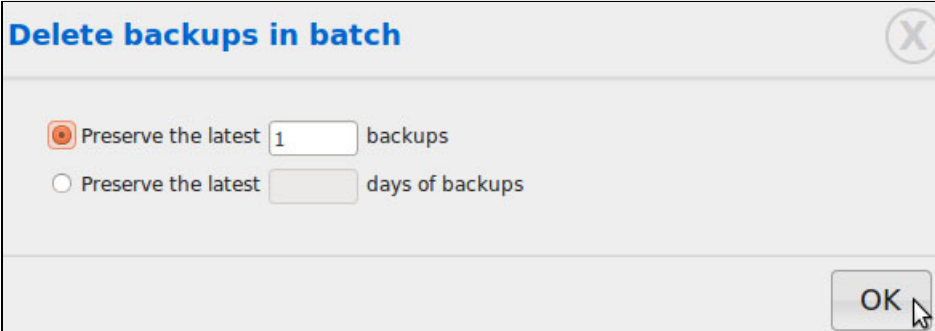
Figure 3-56 shows how to create a new custom backup. Based on Figure 3-55 on page 98, click **New Custom Backup**, enter a simple description in the Description field, enter the paths to the directories to be either included or excluded, then click **OK**.



The screenshot shows a dialog box titled "Create a backup" with a close button (X) in the top right corner. Inside the dialog, there is a "Description" field containing the text "MyBackup". Below this, there are two sections: "Path Included" and "Path Excluded". The "Path Included" section has two input fields, one containing "/etc" and the other containing "/var/log", each with a "+" button to its right. The "Path Excluded" section has one input field containing "/etc/init" with a "+" button to its right. At the bottom right of the dialog is an "OK" button with a mouse cursor pointing at it.

Figure 3-56 Creating a custom backup

Figure 3-57 shows how to delete old backups. It is possible to preserve a latest number of backups or to preserve backups created after a given date.



The screenshot shows a dialog box titled "Delete backups in batch" with a close button (X) in the top right corner. Inside the dialog, there are two radio button options. The first option, "Preserve the latest 1 backups", is selected and has a text input field with the number "1". The second option is "Preserve the latest days of backups", which is unselected and has an empty text input field. At the bottom right of the dialog is an "OK" button with a mouse cursor pointing at it.

Figure 3-57 Removing old backups

## Network Configuration

Figure 3-58 shows the network configuration tool. Click **Edit** to make any field editable and **Save** to persist the changes.

The screenshot shows the 'Network Configuration' tool. It features a table for network interfaces with columns for Name, IP, and Mask. The 'Interface' section lists interfaces like enP1p12s0f0, enP1p12s0f1, enP1p12s0f3, lo, net0, virbr0, virbr0-nic, virbr1, virbr1-nic, and vnet0. The 'DNS' section shows a value of 9.0.128.50, and the 'Gateway' section shows 9.40.193.1. Each interface row has an 'Edit' icon (a square with a pencil) to its right.

Interface	Name	IP	Mask
	enP1p12s0f0		
	enP1p12s0f1		
	enP1p12s0f3		
	lo	127.0.0.1	255.0.0.0
	net0	9.40.193.34	255.255.255.0
	virbr0	192.168.122.1	255.255.255.0
	virbr0-nic		
	virbr1	192.168.121.1	255.255.255.0
	virbr1-nic		
	vnet0		

**DNS**  
9.0.128.50

**Gateway**  
9.40.193.1

Figure 3-58 Network configuration tool

## Power Options tool

Ginger offers a Power Options tool to make it easy to change tuning profiles.

**Note:** It is not possible to create profiles using Ginger.

Figure 3-59 shows the Power Options tool.

The screenshot shows the 'Power Options' tool. It has a legend at the top: 'Activated Option' (checked), 'Selected Option' (radio), and 'Unselected Option' (radio). Below the legend, there are three columns of radio button options. The first column includes 'balanced' (checked), 'desktop-powersave', 'laptop-battery-powersave', 'network-throughput', 'spindown-disk', and 'virtual-host'. The second column includes 'default', 'enterprise-storage', 'latency-performance', 'powersave', and 'throughput-performance'. The third column includes 'desktop', 'laptop-ac-powersave', 'network-latency', 'server-powersave', and 'virtual-guest'. An 'Activate' button is located in the top right corner.

✓ Activated Option   Selected Option   Unselected Option   **Activate**

✓ balanced   default   desktop  
○ desktop-powersave   enterprise-storage   laptop-ac-powersave  
○ laptop-battery-powersave   latency-performance   network-latency  
○ network-throughput   powersave   server-powersave  
○ spindown-disk   throughput-performance   virtual-guest  
○ virtual-host

Figure 3-59 Power Options tool

Figure 3-60 shows how to change a profile. Click **default** for instance, then click **Activate** and the system will be tuned to that particular profile.

This screenshot is identical to Figure 3-59, showing the 'Power Options' tool. A mouse cursor is pointing at the 'Activate' button in the top right corner, indicating the next step in the process of changing a profile.

✓ Activated Option   Selected Option   Unselected Option   **Activate**

✓ balanced   default   desktop  
○ desktop-powersave   enterprise-storage   laptop-ac-powersave  
○ laptop-battery-powersave   latency-performance   network-latency  
○ network-throughput   powersave   server-powersave  
○ spindown-disk   throughput-performance   virtual-guest  
○ virtual-host

Figure 3-60 Changing a profile

## SAN adapters

Storage area network (SAN) connected to a PowerKVM system can be seen in the SAN Adapters pane. Figure 3-61 shows all SAN adapters connected to the system with the following fields displayed:

- ▶ Name
- ▶ Worldwide port name (WWPN)
- ▶ Worldwide node name (WWNN)
- ▶ State
- ▶ In-Use/Max Ports
- ▶ Speed
- ▶ Symbolic names

SAN Adapters					
Name	WWPN	WWNN	State	In-Use/Max Ports	Speed
host2	0x10000090fa02d5fa	0x20000090fa02d5fa	Online	0/895	10 Gbit
host3	0x10000090fa02d5fb	0x20000090fa02d5fb	Online	0/895	10 Gbit
Symbolic Name					
Emulex 00E9267 FV10.2.252.1905 DV10.4.8000.0. HN:flykvm.isst.aus.stglabs.ibm.com OS:Linux					
Emulex 00E9267 FV10.2.252.1905 DV10.4.8000.0. HN:flykvm.isst.aus.stglabs.ibm.com OS:Linux					

Figure 3-61 Listing SAN adapters connected to the system

## Sensor Monitor tool

Figure 3-62 shows the Sensor Monitor tool. It allows users to follow up the host sensors, such as the CPU temperature.

Sensor Monitor	
ibmpowernv-isa-0000	hdds
Core 8-15: 51 C	
Core 16-23: 56 C	
Core 24-31: 50 C	
Core 32-39: 48 C	
Core 40-47: 49 C	
Core 48-55: 50 C	
Core 56-63: 50 C	
Core 64-71: 50 C	
Core 72-79: 49 C	
Centaur 0: 39 C	
Centaur 1: 42 C	

Figure 3-62 Sensor monitor

## SEP Configuration

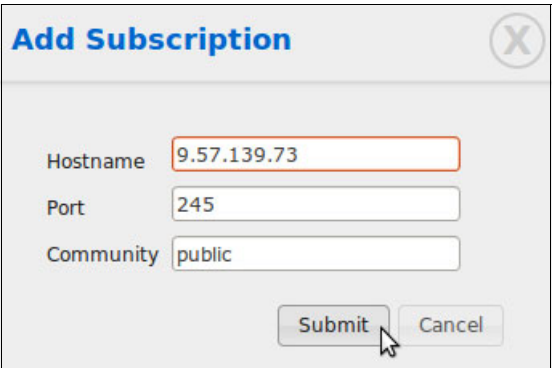
IBM Service Event Provider (SEP) is a service installed in PowerKVM to identify hardware problems and send reports to registered listeners.

Figure 3-63 shows how to start SEP service.

SEP Configuration			
Status:	<input type="radio"/>	Start	
Hostname		Port	Community

Figure 3-63 Starting SEP service

Figure 3-64 shows how to subscribe a listener to SEP. Click **Add**, enter the listener host name in the Hostname field, type the port number at which the listener receives SNMP traps in the Port field, add the SNMP community name in the Community field, and click **Submit**.



The 'Add Subscription' dialog box contains three input fields: 'Hostname' with the value '9.57.139.73', 'Port' with the value '245', and 'Community' with the value 'public'. At the bottom are 'Submit' and 'Cancel' buttons. A mouse cursor is pointing at the 'Submit' button.

Figure 3-64 Subscribing a new listener

Figure 3-65 shows all listeners registered to listen to SNMP traps. It is possible to remove any listener by clicking the **trash can** icon.

SEP Configuration

Status:  Stop

Hostname	Port	Community	
9.57.139.73	245	public	<div>+</div> <div>trash can</div>

Figure 3-65 SEP configuration pane

## User Management tool

Creating and removing users on the host are easily performed with the User Management tool by Ginger. Figure 3-66 shows how to create a new user in the system. Click **Add** to open the Add User dialog box.

User Management

Name	Group	Profile	
nfsnobody	nfsnobody	kimchiuser	<div>+</div> <div>trash can</div>

Figure 3-66 Listing existing users on the host

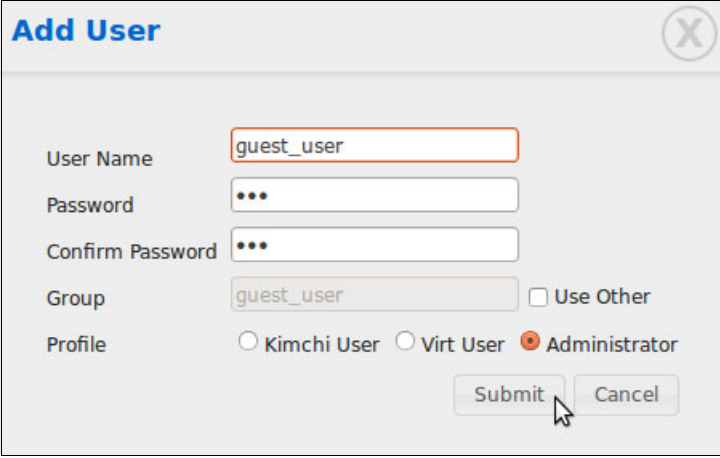
Figure 3-67 on page 103 shows how to add a new user on the host system.

Enter the user name in the User Name field, then enter the password and the password confirmation in the Password and Confirm Password fields respectively, select **Use Other** to edit the Group field if necessary, select the user Profile, and then click **Submit**.

The profile determines the user authorization level on the host system:

- ▶ Kimchi User: A user created to access Kimchi only, that user cannot access the system.
- ▶ Virt User: A regular user added in the KVM group.
- ▶ Administrator: A user added in KVM group with administration privileges.

**Note:** Regular system users are also listed as kimchiuser.



The image shows a web-based dialog box titled "Add User" with a close button (X) in the top right corner. The form contains the following fields and options:

- User Name:** A text input field containing "guest\_user".
- Password:** A password input field with three dots indicating masked text.
- Confirm Password:** A password input field with three dots indicating masked text.
- Group:** A dropdown menu showing "guest\_user". To its right is a checkbox labeled "Use Other".
- Profile:** Three radio button options: "Kimchi User", "Virt User", and "Administrator". The "Administrator" option is selected, indicated by a red dot.
- Buttons:** "Submit" and "Cancel" buttons are located at the bottom right of the form.

Figure 3-67 Adding a new user







## Managing guests from the command-line interface

This chapter describes how to manage IBM PowerKVM virtual machines from the command-line interface (CLI) by using the **virsh** command. It covers the most common commands and options used to manage guests. See the **virsh** command man pages and Libvirt online documentation for detailed information about how to manage guests from the command line.

## 4.1 virsh console

**virsh** is a command-line interface to libvirt (a library to manage the KVM hypervisor). You can manage libvirt from the local host or remotely. If you manage PowerKVM from a remote Linux-based operating system, you must have **virsh** installed on the remote system. You can connect to PowerKVM by using the **virsh connect** command. As an argument, it accepts a URI such as `qemu+ssh://root@hostname/system`, as shown in Example 4-1.

*Example 4-1 Connect remotely*

---

```
% virsh
Welcome to virsh, the virtualization interactive terminal.

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

virsh # connect qemu+ssh://root@powerkvm.itso.ibm.com/system
root@powerkvm.itso.ibm.com's password:

virsh # list
  Id   Name                               State
-----
  8    MyGuest                           running
```

---

It is also possible to log in to PowerKVM through a Secure Shell (SSH) and run **virsh** directly on the target host.

You can run **virsh** in either of two modes:

- Interactive terminal mode, as shown in Example 4-2.

*Example 4-2 Working within virsh interactive shell*

---

```
% virsh
Welcome to virsh, the virtualization interactive terminal.

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

virsh # list
  Id   Name                               State
-----
  8    MyGuest                           running
```

---

- Non-interactive virsh subcommand from a shell prompt, as shown in Example 4-3.

*Example 4-3 Running virsh commands from system shell*

---

```
# virsh list --all
  Id   Name                               State
-----
  8    MyGuest                           running
```

---

### 4.1.1 virsh vncdisplay

If your running guest has a virtual graphical adapter attached, you can connect to a graphical console by using Virtual Network Computing (VNC). You can discover which VNC port a guest is running on with the **virsh vncdisplay** command, as shown in Example 4-4.

*Example 4-4 vncdisplay command*

---

```
# virsh vncdisplay MyGuest
127.0.0.1:1
```

---

## 4.2 Managing storage pools

A *pool* or *storage pool* is a collection of disks that contain all of the data for a specified set of volumes. The pool of storage is created by administrators for use by guests. In other words, storage pools consist of volumes used by guests as block devices. This section describes how to manage storage pools by using the command line. See 6.3, “Storage pools” on page 167 for more information about the storage pool concept.

### 4.2.1 Create new storage pools

Storage pools can be created from a pre-created file with the **virsh pool-create** command or by using the **virsh** command directly:

**virsh pool-create-as** <arguments>

#### Create file-based pools

Example 4-5 shows how to create file-based pools for DIR and NFS.

*Example 4-5 Create a file-based pool*

---

#### DIR:

```
# mkdir /MyPool
# virsh pool-create-as MyPool dir --target=/MyPool
Pool MyPool created
```

```
# virsh pool-list
```

Name	State	Autostart
default	active	no
ISO	active	no
MyPool	active	no

```
# virsh pool-dumpxml MyPool
```

```
<pool type='dir'>
  <name>MyPool</name>
  <uuid>8e2b0942-7831-4f4e-8ace-c5539e3f22d2</uuid>
  <capacity unit='bytes'>21003583488</capacity>
  <allocation unit='bytes'>2164170752</allocation>
  <available unit='bytes'>18839412736</available>
  <source>
  </source>
  <target>
    <path>/MyPool</path>
```

```

    <permissions>
      <mode>0755</mode>
      <owner>0</owner>
      <group>0</group>
      <label>unconfined_u:object_r:default_t:s0</label>
    </permissions>
  </target>
</pool>

```

#### NFS:

```

# mkdir /MyPoolNFS
# virsh pool-create-as MyPoolNFS netfs \
--source-host=9.57.139.73 \
--source-path=/var/www/powerkvm \
--target=/MyPoolNFS
Pool MyPoolNFS created

```

```

# virsh pool-list

```

Name	State	Autostart
default	active	no
ISO	active	no
MyPool	active	no
MyPoolNFS	active	no

```

# virsh pool-dumpxml MyPoolNFS
<pool type='netfs'>
  <name>MyPoolNFS</name>
  <uuid>a566ca99-a64c-4793-8c7f-d394aee058ef</uuid>
  <capacity unit='bytes'>197774016512</capacity>
  <allocation unit='bytes'>72522661888</allocation>
  <available unit='bytes'>125251354624</available>
  <source>
    <host name='9.57.139.73' />
    <dir path='/var/www/powerkvm' />
    <format type='auto' />
  </source>
  <target>
    <path>/MyPoolNFS</path>
    <permissions>
      <mode>0755</mode>
      <owner>0</owner>
      <group>0</group>
      <label>system_u:object_r:nfs_t:s0</label>
    </permissions>
  </target>
</pool>

```

## Create block-based pools

Block-based pools are *logical*, *iSCSI*, and *SCSI* types. Example 4-6 on page 109 shows how to create them.

**Note:** The LVM2-based pool must be created with the **pool-define-as** command and later built and activated.

**LVM2:**

```
# virsh pool-define-as MyPoolLVM logical \
--source-dev /dev/sdb1 \
--target /dev/MyPoolLVM
Pool MyPoolLVM defined
```

```
# virsh pool-build MyPoolLVM
Pool MyPoolLVM built
```

```
# virsh pool-start MyPoolLVM
Pool MyPoolLVM started
```

```
# virsh pool-list
Name                State      Autostart
-----
default             active     no
ISO                  active     no
MyPoolLVM            active     no
```

```
# virsh pool-dumpxml MyPoolLVM
<pool type='logical'>
  <name>MyPoolLVM</name>
  <uuid>37ac662e-28ab-4f80-a981-c62b20966e1e</uuid>
  <capacity unit='bytes'>0</capacity>
  <allocation unit='bytes'>0</allocation>
  <available unit='bytes'>0</available>
  <source>
    <device path='/dev/sdb1'>
      <name>MyPoolLVM</name>
      <format type='lvm2'>
    </source>
  <target>
    <path>/dev/MyPoolLVM</path>
  </target>
</pool>
```

```
# vgs
VG                #PV #LV #SN Attr   VSize  VFree
MyPoolLVM         1  0  0 wz--n- 931.51g 931.51g
```

**iSCSI:**

```
# virsh pool-create-as MyPoolISCSI iscsi \
--source-host 9.40.193.34 \
--source-dev iqn.1986-03.com.ibm.2145:mypooliscsi \
--target /dev/disk/by-id
Pool MyPoolISCSI created

# virsh vol-list MyPoolISCSI
Name          Path
-----
unit:0:0:1    /dev/disk/by-id/wwn-0x6000000000000000e0000000010001

# virsh pool-list
Name          State      Autostart
-----
default       active    no
ISO           active    no
MyPoolISCSI   active    no

# virsh pool-dumpxml MyPoolISCSI
<pool type='iscsi'>
  <name>MyPoolISCSI</name>
  <uuid>f4f5daf2-13a1-4781-b3e6-851d9435416c</uuid>
  <capacity unit='bytes'>107374182400</capacity>
  <allocation unit='bytes'>107374182400</allocation>
  <available unit='bytes'>0</available>
  <source>
    <host name='9.40.193.34' port='3260'>/>
    <device path='iqn.1986-03.com.ibm.2145:mypooliscsi'>/>
  </source>
  <target>
    <path>/dev/disk/by-id</path>
  </target>
</pool>
```

---

## 4.2.2 Query available storage pools

To list available storage pools, run the **pool-list** command, as shown in Example 4-7.

*Example 4-7 Query available storage pools*

---

```
# virsh pool-list
Name          State      Autostart
-----
default       active    no
ISO           active    no
MyPoolISCSI   active    no
```

---

To list details of a specific storage pool, run **pool-info**, as in Example 4-8 on page 111.

#### Example 4-8 Display pool information

```
# virsh pool-info default
Name:          default
UUID:          078e187f-6838-421e-b3e7-7f5c515867b8
State:         running
Persistent:    yes
Autostart:     no
Capacity:      878.86 GiB
Allocation:    3.21 GiB
Available:     875.65 GiB
```

**Tip:** To avoid using two commands, you can use the `--details` flag shown in Example 4-9.

#### Example 4-9 Display a verbose pool list

```
# virsh pool-list --details
```

Name	State	Autostart	Persistent	Capacity	Allocation	Available
default	running	no	yes	878.86 GiB	3.21 GiB	875.65 GiB
ISO	running	no	yes	9.72 GiB	161.52 MiB	9.56 GiB
MyPoolISCSI	running	no	no	100.00 GiB	100.00 GiB	0.00 B

## 4.2.3 List available volumes

To list created volumes in a particular storage pool, use the `vol-list` command. The storage pool must be specified as an argument. Example 4-10 shows how you can display volumes.

#### Example 4-10 Display volume list

```
# virsh vol-list MyPoolISCSI
```

Name	Path
unit:0:0:1	/dev/disk/by-id/wwn-0x6000000000000000e0000000010001

## 4.2.4 Create a new volume

To create a new volume in a particular storage pool, use the `vol-create-as` command, as shown in Example 4-11.

#### Example 4-11 Create volume in storage pool

```
# virsh vol-create-as MyPool mypool.qcow2 --format qcow2 30G --allocation 4G
Vol mypool.qcow2 created
```

In this example, a 30 GB volume named *mypool.qcow2* is created in the pool named *MyPool*. The `--allocation` argument instructs libvirt to allocate only 4 GB. The rest will be allocated on demand. This is sometime referred to as a *thin volume*.

Example 4-12 shows how to check whether the volume has been created.

*Example 4-12 Display the volume list*

```
# virsh vol-list MyPool
Name                Path
-----
mypool.qcow2        /MyPool/mypool.qcow2
```

If you need a more verbose output, use the **--details** flag as shown in Example 4-13.

*Example 4-13 Display verbose volume list*

```
# virsh vol-list MyPool --details
Name                Path                Type    Capacity  Allocation
-----
mypool.qcow2        /MyPool/mypool.qcow2  file    30.00 GiB  196.00 KiB
```

## 4.2.5 Delete or wipe a volume

To wipe a volume from a pool, you can use the **vol-wipe** command shown in Example 4-14.

*Example 4-14 Wipe volume*

```
# virsh vol-wipe mypool.qcow2 MyPool
Vol mypool.qcow2 wiped

# virsh vol-list MyPool --details
Name                Path                Type    Capacity  Allocation
-----
mypool.qcow2        /MyPool/mypool.qcow2  file    196.00 KiB  196.00 KiB
```

The wiped volumes are empty and can be reused for another guest.

To remove a volume completely, use the **vol-delete** command, as shown in Example 4-15.

*Example 4-15 Volume delete*

```
# virsh vol-delete mypool.qcow2 MyPool
Vol mypool.qcow2 deleted
```

**Note:** This command deletes the volume and therefore the data on the volume.

## 4.2.6 Snapshots

Snapshots save the current machine state (disk, memory, and device states) to be used later. It is specially interesting if a user is going to perform actions that can destroy data. In this case, a snapshot taken before that destructive operation can be reverted and the guest will continue working from that point.

**Note:** virsh snapshot-revert loses all changes made in the guest since the snapshot was taken.



Example 4-16 shows how to create, list, revert, and delete snapshots.

*Example 4-16 Working with snapshots*

---

```
# virsh snapshot-create-as PowerKVM_VirtualMachine
Domain snapshot 1447164760 created

# virsh snapshot-create-as PowerKVM_VirtualMachine MyNewSnapshot
Domain snapshot MyNewSnapshot created

# virsh snapshot-revert PowerKVM_VirtualMachine MyNewSnapshot

# virsh snapshot-current PowerKVM_VirtualMachine --name
MyNewSnapshot

# virsh snapshot-create-as PowerKVM_VirtualMachine NewChild
Domain snapshot NewChild created

# virsh snapshot-list PowerKVM_VirtualMachine --parent
Name                Creation Time        State      Parent
-----
1447164760          2015-11-10 09:12:40 -0500 shutoff    (null)
MyNewSnapshot        2015-11-10 09:14:20 -0500 shutoff    1447164760
NewChild             2015-11-10 09:22:23 -0500 shutoff    MyNewSnapshot

# virsh snapshot-delete PowerKVM_VirtualMachine MyNewSnapshot
Domain snapshot MyNewSnapshot deleted
```

---

When MyNewSnapshot was deleted, its content was merged into NewChild. Figure 4-1 illustrates what happens to a child when its parent snapshot is deleted.

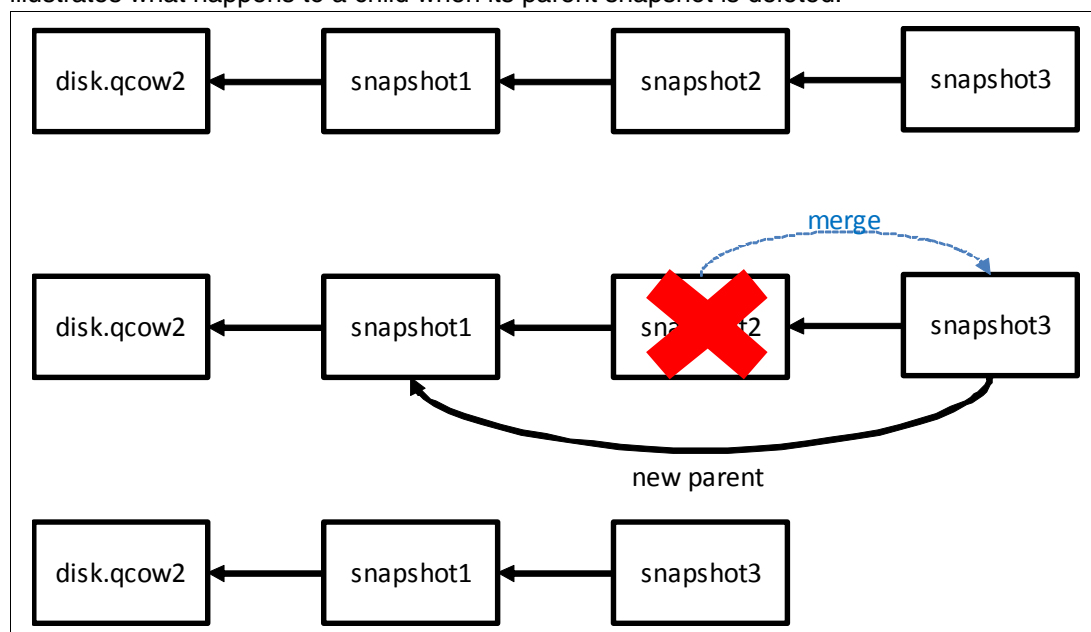


Figure 4-1 Deleting a snapshot

## 4.3 Manage guest networks

This section describes how to manage guest networks from the command line.

The hypervisor network configuration is also described as an XML file. The files that describe the host network configuration are stored in the `/var/lib/libvirt/network` directory.

The default configuration that is based on Network Address Translation (NAT) is called the libvirt *default* network. After the PowerKVM installation, the default network is automatically created and available for use.

When **libvirt** adds a network definition based on a bridge, the bridge is created automatically. You can see it by using the **brctl show** command, as shown in Example 4-17.

*Example 4-17 Command to show the bridge interfaces on the system*

---

```
# brctl show
bridge name      bridge id        STP enabled    interfaces
virbr0           8000.5254000ec7c6  yes           virbr0-nic
                  vnet1
virbr1           8000.525400dd81a0  yes           virbr1-nic
```

---

### 4.3.1 Query guest networks

To list defined guest networks, use the **net-list** command shown in Example 4-18.

*Example 4-18 Display networks*

---

```
# virsh net-list --all
Name              State    Autostart    Persistent
-----
default           active   yes          yes
kop               active   yes          yes
```

---

To list detailed information about a specific network entry, use the **net-info** command, as shown in Example 4-19.

*Example 4-19 Display specific network*

---

```
# virsh net-info default
Name:             default
UUID:             b7adb719-2cd9-4c7a-b908-682cdd4e0f8a
Active:           yes
Persistent:       yes
Autostart:        yes
Bridge:           virbr0
```

---

### 4.3.2 Create a guest network

To create a guest network, use the **virsh net-create** command to create a transitory network or use the **virsh net-define** command to create persistent network, that will last after shutdowns. That command requires a pre-created XML file with a network definition. See Example 4-20, Example 4-21 on page 116, and Example 4-22 on page 116 for Bridged, NAT, and Open vSwitch networks, respectively. For more information about a host network, see 6.2, “Network virtualization” on page 164.

In the next examples, we use the **uuidgen** command to create unique identifiers for those devices. However, libvirt can generate it automatically if the **uuid** parameter is missed.

*Example 4-20 NAT definition*

---

```
# uuidgen
a2c0da29-4e9c-452a-9b06-2bbe9d8f8f65

# cat nat.xml
<network>
  <name>MyNat</name>
  <uuid>a2c0da29-4e9c-452a-9b06-2bbe9d8f8f65</uuid>
  <forward mode='nat' />
  <bridge name='myvirbr0' stp='on' delay='0' />
  <ip address='192.168.133.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.133.2' end='192.168.133.50' />
    </dhcp>
  </ip>
</network>

# virsh net-define nat.xml
Network MyNat defined from nat.xml

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

```
# virsh net-list
```

Name	State	Autostart	Persistent
bridge	active	yes	yes
default	active	yes	yes
kop	active	yes	yes
MyNat	active	no	yes

---

#### Example 4-21 Bridged network

---

```
# uuidgen
8ee4536b-c4d3-4e3e-a139-6108f3c2d5f5

# cat brdg.xml
<network>
  <name>MyBridge</name>
  <uuid>8ee4536b-c4d3-4e3e-a139-6108f3c2d5f5</uuid>
  <forward dev='enP1p12s0f0' mode='bridge'>
    <interface dev='enP1p12s0f0' />
  </forward>
</network>

# virsh net-define brdg.xml
Network MyBridge defined from brdg.xml

# virsh net-start MyBridge
Network MyBridge started

# virsh net-list
  Name                State      Autostart   Persistent
  -----
  bridge              active     yes         yes
  default             active     yes         yes
  kop                 active     yes         yes
  MyBridge            active     no          yes
  MyNat              active     no          yes
```

---

#### Example 4-22 Open vSwitch configuration

---

```
# systemctl start openvswitch
# ovs-vsctl add-br myOVS
# ovs-vsctl add-port myOVS enP1p12s0f1
# ovs-vsctl show
ed0a4b3d-0738-468e-9642-0282c4342960
    Bridge myOVS
        Port myOVS
            Interface myOVS
                type: internal
        Port "enP1p12s0f1"
            Interface "enP1p12s0f1"

# cat ovs.xml
<network>
  <name>MyOVSBrc</name>
  <forward mode='bridge' />
  <bridge name='myOVS' />
  <virtualport type='openvswitch' />
  <portgroup name='default' default='yes'>
  </portgroup>
</network>

# virsh net-define ovs.xml
Network MyOVSBrc defined from ovs.xml
```

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

```
# virsh net-list
```

Name	State	Autostart	Persistent
bridge	active	yes	yes
default	active	yes	yes
kop	active	yes	yes
MyBridge	active	no	yes
MyNat	active	no	yes
MyOVSB	active	no	yes

Example 4-23 shows what the guest network interface looks like when using the Open vSwitch bridge.

*Example 4-23 Open vSwitch bridge interface*

```
# virsh dumpxml PowerKVM_VirtualMachine
```

```
...
<interface type='bridge'>
  <mac address='52:54:00:c9:e4:99' />
  <source network='MyOVSB' bridge='myOVS' />
  <virtualport type='openvswitch'>
    <parameters interfaceid='41c249bb-c59d-4632-9601-25d0a9285755' />
  </virtualport>
  <target dev='vnet0' />
  <model type='virtio' />
  <alias name='net0' />
  <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x0' />
</interface>
...
```

## 4.4 Managing guests

This section describes how to manage guests from the command line. The guest configuration is represented as an XML file by libvirt.

### 4.4.1 Create a new guest

To create new guests from command line, use the **virt-install** command. The following are the core arguments:

<b>--name</b>	Defines a guest name
<b>--vcpus</b>	Defines the number of CPUs used by the guest
<b>--memory</b>	Defines the amount of RAM, in megabytes, for a guest
<b>--cdrom</b>	Defines the ISO that is used for guest installation
<b>--disk</b>	Defines a block device for a guest
<b>--network</b>	Specifies a network to plug in to a guest

Each option has a subset of options. Only the most important are covered in this chapter.

The **vcpus** argument has the following options (also see Example 4-24):

<b>maxvcpus</b>	Sets the maximum number of CPUs that you can hot plug into a guest
<b>threads</b>	Sets the number of threads used by a guest

*Example 4-24 vcpus options*

---

```
--vcpus=2,threads=8,maxvcpus=5
```

---

The **disk** argument has the following suboptions, which are also shown in Example 4-25.

<b>pool</b>	Specifies the pool name where you are provisioning the volume
<b>size</b>	Specifies the volume size in GB
<b>format</b>	Format of resulting image (raw   qcow2)
<b>bus</b>	Bus used by the block device

*Example 4-25 Disk options*

---

```
--disk pool=default,size=10,format=qcow2,bus=spapr-scsi
```

---

Example 4-26 shows an example of attaching a volume from a Fibre Channel or iSCSI pool.

*Example 4-26 LUN mapping*

---

```
--disk vol=poolname/unit:0:0:1
```

---

The **network** argument has suboptions, the most important are:

<b>bridge</b>	Host bridge device
<b>network</b>	Network name created in “Create a guest network” on page 115

Example 4-27 shows examples of networks.

*Example 4-27 Network arguments*

---

```
--network bridge=myvirbr0  
--network default,mac=52.54.00.bd.7f.d5  
--network network=MyOSVBr
```

---

Example 4-28 shows the full command and argument.

*Example 4-28 virt-install example*

---

```
# virt-install --name PowerKVM_VirtualMachine \  
--memory 4096 \  
--vcpus=2,threads=2,maxvcpus=8 \  
--cdrom /var/lib/libvirt/images/distro.iso \  
--disk pool=default,size=100,sparse=true,cache=none,format=qcow2 \  
--network default
```

---

The **virt-install** command automatically starts the installation and attaches the console.

## 4.4.2 List guests

By default, the **list** command displays only running guests. If you want to see powered off guests, use the **--all** flag, as shown in Example 4-29 on page 119.

*Example 4-29 List all guests*

---

```
# virsh list --all
 Id      Name                                State
-----
 8       MyGuest                             running
-       PowerKVM_VirtualMachine           shut off
```

---

### 4.4.3 Start or stop a guest

To start an existing guest, use the **start** command as in Example 4-30.

*Example 4-30 Start VM*

---

```
# virsh start PowerKVM_VirtualMachine
Domain PowerKVM_VirtualMachine started
```

---

To power off the guest, use the **destroy** or the **shutdown** command as in Example 4-31. The shutdown command interacts with the guest operating system to shut down the system gracefully. This operation can take some time because all services must be stopped. The destroy command shutdown the guest immediately. It can damage the guest operating system.

*Example 4-31 Stop/halt VM*

---

```
# virsh list --all
 Id      Name                                State
-----
 8       MyGuest                             running
25       PowerKVM_VirtualMachine           running
```

```
# virsh destroy PowerKVM_VirtualMachine
Domain PowerKVM_VirtualMachine destroyed
```

```
# virsh list --all
 Id      Name                                State
-----
 8       MyGuest                             running
-       PowerKVM_VirtualMachine           shut off
```

```
# virsh list --all
 Id      Name                                State
-----
 8       MyGuest                             running
28       PowerKVM_VirtualMachine           running
```

```
# virsh shutdown PowerKVM_VirtualMachine
Domain PowerKVM_VirtualMachine is being shutdown
```

```
# virsh list --all
 Id      Name                                State
-----
 8       MyGuest                             running
-       PowerKVM_VirtualMachine           shut off
```

---

## 4.4.4 Suspending and resuming

A virtual machine can be suspended. This feature pauses the guest and stops using CPU resources until it is resumed. However, it still uses host memory. A guest can be paused by using the **suspend** command. After a guest is suspended, it is changed to the paused state and can be resumed by using the **resume** command.

Example 4-32 shows the state of the guest before it is suspended, after it is suspended, and after it is resumed.

*Example 4-32 Suspending and resuming a guest*

---

```
# virsh list
  Id      Name                                State
-----
  60      PowerKVM_VirtualMachine              running

# virsh suspend PowerKVM_VirtualMachine
Domain PowerKVM_VirtualMachine suspended

# virsh list
  Id      Name                                State
-----
  60      PowerKVM_VirtualMachine              paused

# virsh resume PowerKVM_VirtualMachine
Domain PowerKVM_VirtualMachine resumed

# virsh list
  Id      Name                                State
-----
  60      PowerKVM_VirtualMachine              running
```

---

## 4.4.5 Delete a guest

To delete a guest, run the **virsh undefine** command as shown in Example 4-33.

*Example 4-33 Deleting a guest*

---

```
# virsh undefine PowerKVM_VirtualMachine
Domain PowerKVM_VirtualMachine has been undefined
```

---



## 4.4.6 Connect to a guest

To connect to an already running guest, use the **console** command as shown in Example 4-34.

*Example 4-34 Console access*

---

```
# virsh start PowerKVM_VirtualMachine
Domain PowerKVM_VirtualMachine started

# virsh console PowerKVM_VirtualMachine
Connected to domain PowerKVM_VirtualMachine
Escape character is ^]
```

---

**Note:** To detach an open console, hold down the Ctrl key and press the ] key.

It is also possible to start a guest and attach to the console by using one command:

```
# virsh start PowerKVM_VirtualMachine --console
```

## 4.4.7 Edit a guest

The **virsh edit** command is used to edit any guest parameter. It does not provide a friendly user interface such as Kimchi. In contrast, virsh opens the guest XML in a text editor where changes must be done manually.

Example 4-35 shows how to edit a guest. Regardless of using a common editor, virsh verifies any possible mistake to avoid guest damage.

*Example 4-35 Editing guest configuration*

---

```
# virsh edit PowerKVM_VirtualMachine
<domain type='kvm'>
  <name>PowerKVM_VirtualMachine</name>
  <uuid>6009664b-27c5-4717-97c5-370917c9594f</uuid>
  <memory unit='KiB'>4194304</memory>
  <currentMemory unit='KiB'>4194304</currentMemory>
  <vcpu placement='static' current='2'>8</vcpu>
  <os>
    <type arch='ppc64le' machine='pseries-2.4'>hvm</type>
    <boot dev='hd'>/>
  </os>
  <features>
    <acpi/>
    <apic/>
  </features>
  <cpu>
    <topology sockets='4' cores='1' threads='2'>/>
  </cpu>
  ...
```

---

After quitting the editor, virsh checks the file for errors and returns one of the following messages:

- Domain guest XML configuration not changed

- ▶ Error: operation failed... Try again? [y,n,i,f,?]
- ▶ Domain guest XML configuration edited

The default text editor used by virsh is vi, but it is possible to use any other editor by setting the EDITOR shell variable. Example 4-36 shows how to configure a different text editor for virsh.

*Example 4-36 Using another text editor*

---

```
EDITOR=nano virsh edit PowerKVM_VirtualMachine
```

---

## 4.4.8 Add new storage to an existing guest

Example 4-37 shows how to create a new virtual disk, how to plug that disk into the guest, and how to unplug the disk.

*Example 4-37 Adding new virtual storage*

---

```
# qemu-img create -f qcow2 mynewdisk.qcow2 80G
Formatting 'mynewdisk.qcow2', fmt=qcow2 size=85899345920 encryption=off
cluster_size=65536 lazy_refcounts=off refcount_bits=16

# cat disk.xml
<disk type='file' device='disk'>
  <driver name='qemu' type='qcow2' />
  <source file='/var/lib/libvirt/images/mynewdisk.qcow2' />
  <target dev='vdb' bus='virtio' />
</disk>

# virsh attach-device PowerKVM_VirtualMachine disk.xml --config
Device attached successfully
```

---

The **--config** parameter persists the attached device in the guest XML. It means that the disk lasts after reboots and, if the device is attached while the guest is running, the device will only be attached after a reboot. To unplug, run the following command:

```
# virsh detach-device PowerKVM_VirtualMachine disk.xml --config
```

Example 4-38 shows how to perform a disk hotplug. By using the same disk created in Example 4-37, start the guest and run.

*Example 4-38 Storage live pass-through*

---

```
# virsh attach-device PowerKVM_VirtualMachine disk.xml --live
Device attached successfully

# virsh detach-device PowerKVM_VirtualMachine disk.xml --live
Device detached successfully
```

---

The **--live** parameter attaches the device into a running guest. That will be automatically detached as soon as the guest is turned off.

**Note:** If the guest is running, both the **--live** and **--config** parameters can be used together.

## 4.4.9 Add a new network to an existing guest

To attach a network interface, create the XML with the new virtual card and attach it by using the same **virsh attach-device** command. Example 4-39 shows how to plug and unplug such a device.

*Example 4-39 Network interface pass-through*

---

```
# cat network.xml
<interface type='network'>
  <mac address='52:54:00:ba:00:00' />
  <source network='default' />
  <model type='virtio' />
</interface>

# virsh attach-device PowerKVM_VirtualMachine network.xml --config
Device attached successfully

# virsh detach-device PowerKVM_VirtualMachine network.xml --config
Device detached successfully
```

---

Example 4-40 shows how to hotplug a network interface and how to unplug it.

*Example 4-40 Network interface hotplug*

---

```
# virsh attach-device PowerKVM_VirtualMachine network.xml --live
Device attached successfully

(the command ifconfig -a was run in the guest)
# ifconfig -a
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.122.61 netmask 255.255.255.0 broadcast 192.168.122.255
    inet6 fe80::5054:ff:fecl:c718 prefixlen 64 scopeid 0x20<link>
    ether 52:54:00:c1:c7:18 txqueuelen 1000 (Ethernet)
    RX packets 67 bytes 7095 (6.9 KiB)
    RX errors 0 dropped 9 overruns 0 frame 0
    TX packets 57 bytes 6732 (6.5 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth1: flags=4098<BROADCAST,MULTICAST> mtu 1500
    ether 52:54:00:ba:00:00 txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

# virsh detach-device PowerKVM_VirtualMachine network.xml --live
Device detached successfully
```

---

#### 4.4.10 PCI I/O pass-through

The **lspci** command lists all devices attached to the host. Example 4-41 shows how to list all devices, so we can choose the device to be attached. Refer to 6.4, “I/O pass-through” on page 170 for more details about PCI pass-through.

*Example 4-41 Listing host PCI devices*

---

```
# lspci
0000:00:00.0 PCI bridge: IBM Device 03dc
0001:00:00.0 PCI bridge: IBM Device 03dc
0001:01:00.0 PCI bridge: PLX Technology, Inc. Device 8748 (rev ca)
0001:02:01.0 PCI bridge: PLX Technology, Inc. Device 8748 (rev ca)
0001:02:08.0 PCI bridge: PLX Technology, Inc. Device 8748 (rev ca)
0001:02:09.0 PCI bridge: PLX Technology, Inc. Device 8748 (rev ca)
0001:02:0a.0 PCI bridge: PLX Technology, Inc. Device 8748 (rev ca)
0001:02:10.0 PCI bridge: PLX Technology, Inc. Device 8748 (rev ca)
0001:02:11.0 PCI bridge: PLX Technology, Inc. Device 8748 (rev ca)
0001:08:00.0 SATA controller: Marvell Technology Group Ltd. 88SE9235 PCIe 2.0 x2
4-port SATA 6 Gb/s Controller (rev 11)
0001:09:00.0 USB controller: Texas Instruments TUSB73x0 SuperSpeed USB 3.0 xHCI
Host Controller (rev 02)
0001:0a:00.0 PCI bridge: ASPEED Technology, Inc. AST1150 PCI-to-PCI Bridge (rev
03)
0001:0b:00.0 VGA compatible controller: ASPEED Technology, Inc. ASPEED Graphics
Family (rev 30)
0001:0c:00.0 Ethernet controller: Broadcom Corporation NetXtreme BCM5719 Gigabit
Ethernet PCIe (rev 01)
0001:0c:00.1 Ethernet controller: Broadcom Corporation NetXtreme BCM5719 Gigabit
Ethernet PCIe (rev 01)
0001:0c:00.2 Ethernet controller: Broadcom Corporation NetXtreme BCM5719 Gigabit
Ethernet PCIe (rev 01)
0001:0c:00.3 Ethernet controller: Broadcom Corporation NetXtreme BCM5719 Gigabit
Ethernet PCIe (rev 01)
0002:00:00.0 PCI bridge: IBM Device 03dc
```

---

In Example 4-42, we create the XML with the host PCI address information. This is necessary to attach the device to the guest. In this example, we choose the device *0001:0b:00.0*.

After creating the XML, it is necessary to detach the device from the host. This is achieved by running **virsh nodedev-detach**. Then, the **virsh attach-device** command can be used.

*Example 4-42 Getting PCI device information*

---

```
# cat pci.xml
<hostdev mode='subsystem' type='pci' managed='yes'>
  <driver name='vfio'/>
  <source>
    <address domain='0x0001' bus='0x0b' slot='0x00' function='0x00'/>
  </source>
</hostdev>

# virsh nodedev-detach pci_0001_0b_00_0
Device pci_0001_0b_00_0 detached

# virsh attach-device PowerKVM_VirtualMachine pci.xml --config
```

Device attached successfully

```
# virsh detach-device PowerKVM_VirtualMachine pci.xml --config  
Device detached successfully
```

---

As mentioned, the `--config` parameter means the command has effect after the guest rebooting. For a live action, the `--live` parameter must be set. Figure 4-2 shows how to hotplug a PCI device to a guest.

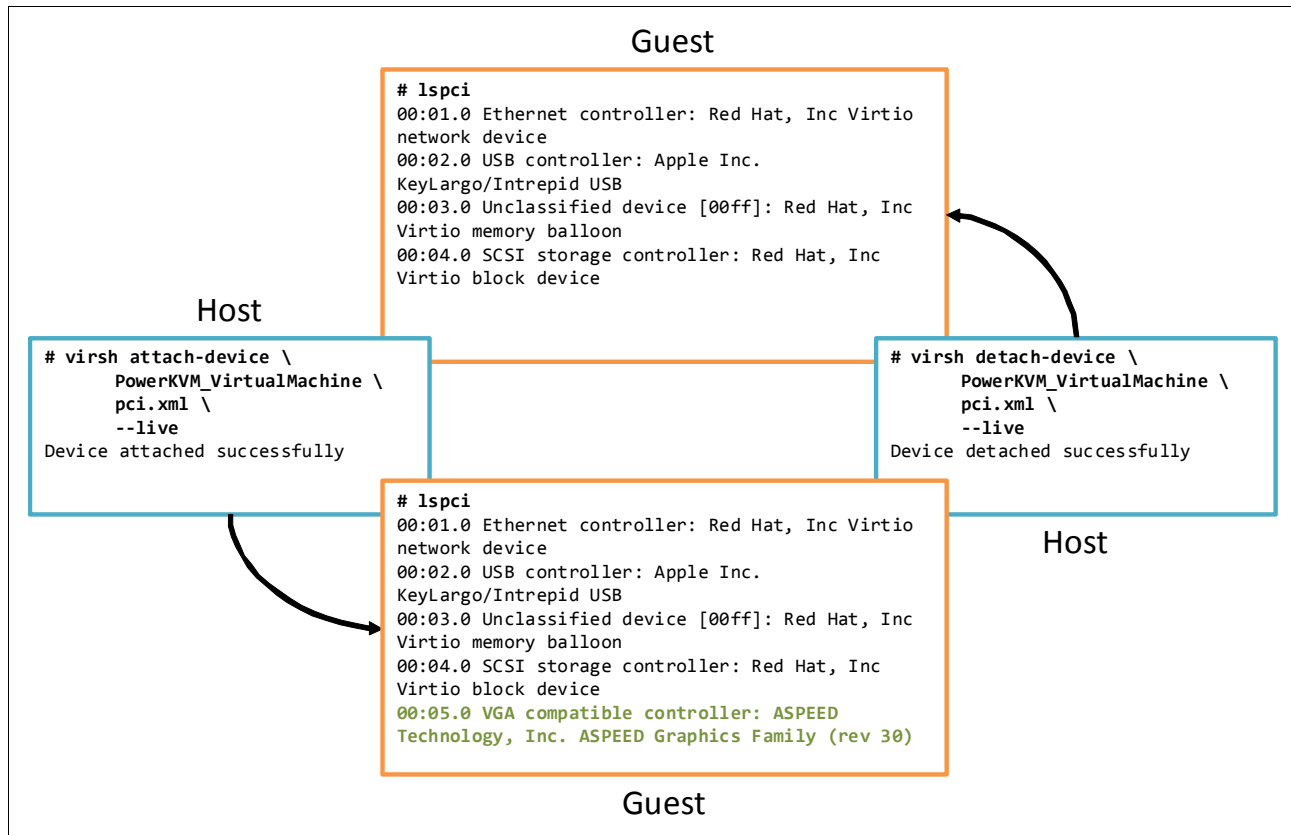


Figure 4-2 Interaction between host and guest during PCI hotplug

When the device is not in use by any guest, it can be reattached to the host by calling the following command:

```
# virsh nodedev-reattach pci_0001_0b_00_0
Device pci_0001_0b_00_0 re-attached
```

When thinking about multi-function PCI pass-through, some rules must be observed:

- ▶ All functions must be detached from the host
- ▶ All functions must be attached to the same guest

**Note:** Multi-function PCI device live pass-through is not currently supported.

Example 4-43 shows how to attach a multifunction device step-by-step. Each function is defined in its own XML, where the guest PCI address is also defined.

**Note:** The first function definition requires the **multifunction='on'** parameter in the guest PCI address.

*Example 4-43 Attaching a multi-function PCI device*

---

```
# cat multif0.xml
<hostdev mode='subsystem' type='pci' managed='yes'>
  <driver name='vfio' />
  <source>
    <address domain='0x0001' bus='0x0c' slot='0x00' function='0x0' />
  </source>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'
multifunction='on' />
</hostdev>

# cat multif1.xml
<hostdev mode='subsystem' type='pci' managed='yes'>
  <driver name='vfio' />
  <source>
    <address domain='0x0001' bus='0x0c' slot='0x00' function='0x1' />
  </source>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x1' />
</hostdev>

# cat multif2.xml
<hostdev mode='subsystem' type='pci' managed='yes'>
  <driver name='vfio' />
  <source>
    <address domain='0x0001' bus='0x0c' slot='0x00' function='0x2' />
  </source>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x2' />
</hostdev>

# cat multif3.xml
<hostdev mode='subsystem' type='pci' managed='yes'>
  <driver name='vfio' />
  <source>
    <address domain='0x0001' bus='0x0c' slot='0x00' function='0x3' />
  </source>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x3' />
</hostdev>

# virsh nodedev-detach pci_0001_0c_00_0
Device pci_0001_0c_00_0 detached

# virsh nodedev-detach pci_0001_0c_00_1
Device pci_0001_0c_00_1 detached

# virsh nodedev-detach pci_0001_0c_00_2
Device pci_0001_0c_00_2 detached

# virsh nodedev-detach pci_0001_0c_00_3
```

```
Device pci_0001_0c_00_3 detached

# virsh attach-device PowerKVM_VirtualMachine multif0.xml --config
Device attached successfully

# virsh attach-device PowerKVM_VirtualMachine multif1.xml --config
Device attached successfully

# virsh attach-device PowerKVM_VirtualMachine multif2.xml --config
Device attached successfully

# virsh attach-device PowerKVM_VirtualMachine multif3.xml --config
Device attached successfully
```

---

Example 4-44 shows how the multifunction device is displayed in the guest.

*Example 4-44 Guest displaying multifunction device*

---

```
# virsh start PowerKVM_VirtualMachine --console

# lspci
00:01.0 Ethernet controller: Red Hat, Inc Virtio network device
00:02.0 USB controller: Apple Inc. KeyLargo/Intrepid USB
00:03.0 Unclassified device [00ff]: Red Hat, Inc Virtio memory balloon
00:04.0 SCSI storage controller: Red Hat, Inc Virtio block device
00:05.0 Ethernet controller: Broadcom Corporation NetXtreme BCM5719 Gigabit Ethernet PCIe (rev 01)
00:05.1 Ethernet controller: Broadcom Corporation NetXtreme BCM5719 Gigabit Ethernet PCIe (rev 01)
00:05.2 Ethernet controller: Broadcom Corporation NetXtreme BCM5719 Gigabit Ethernet PCIe (rev 01)
00:05.3 Ethernet controller: Broadcom Corporation NetXtreme BCM5719 Gigabit Ethernet PCIe (rev 01)
```

---

Example 4-45 shows how to reattach the multifunction device to the host.

*Example 4-45 Reattaching multifunction device to the host*

---

```
# virsh nodedev-reattach pci_0001_0c_00_0
Device pci_0001_0c_00_0 re-attached

# virsh nodedev-reattach pci_0001_0c_00_1
Device pci_0001_0c_00_1 re-attached

# virsh nodedev-reattach pci_0001_0c_00_2
Device pci_0001_0c_00_2 re-attached

# virsh nodedev-reattach pci_0001_0c_00_3
Device pci_0001_0c_00_3 re-attached
```

---

#### 4.4.11 CPU Hotplug

Refer to 5.4, “CPU Hotplug” on page 145.

### 4.4.12 Memory Hotplug

Refer to 5.6, “Memory Hotplug” on page 157.

### 4.4.13 Clone a guest

Example 4-46 shows how to clone a guest by using the **virt-clone** command.

*Example 4-46 Creating a clone guest*

---

```
# virt-clone --original PowerKVM_VirtualMachine \  
--name PowerKVM_Clone \  
--file /var/lib/libvirt/images/clone.qcow2  
Clone 'PowerKVM_Clone' created successfully.
```

```
[root@lxc-hab1 ~]# virsh list --all
```

Id	Name	State
-----		
-	MyGuest	shut off
-	PowerKVM_Clone	shut off
-	PowerKVM_VirtualMachine	shut off

---

### 4.4.14 Migration

Refer to section 7.2, “Guest migration” on page 198.





# Processor and memory virtualization

Machine virtualization involves all of the major server components. Proper configuration and tuning of each component is important to maximize the server utilization. The four major areas of a virtualized system involve CPU, memory, network, and storage.

This chapter covers CPU and memory on IBM PowerKVM and includes the following topics:

- ▶ Resources overcommitment
- ▶ CPU compatibility mode
- ▶ SMT support
- ▶ Dynamic and static Micro-Threading mode
- ▶ CPU pinning
- ▶ CPU shares
- ▶ NUMA
- ▶ Huge pages
- ▶ CPU and memory hotplug

Chapter 6, “I/O virtualization” on page 163 covers the I/O subsystem, which includes networking and storage.

## 5.1 CPU virtualization

*CPU virtualization* is a technique that allows a virtual CPU to run over another CPU (virtual or physical). The process of running a virtual CPU on top of another virtual CPU is called *nested virtualization*, and that topic is outside the scope of this publication. This chapter covers only CPU virtualization over a physical CPU.

In the beginning of CPU virtualization, most of the instructions that ran on the virtual CPU were emulated. But with recent virtualization technologies, most of the guest instructions run directly on the physical CPU, which avoids the translation overhead.

The different ways to virtualize CPUs are covered in the sections that follow.

### 5.1.1 Types of virtualization

When an operating system runs inside a virtual machine, it can work in two different ways, depending on how it interacts with the hypervisor layer: Full virtualization or paravirtualization.

#### Full virtualization

In full virtualization mode, the guest operating system runs inside the virtual machine and does not know that it is running in a virtualized environment. This means that the guest operating system has instructions to run on real hardware, so the hypervisor needs to emulate the real hardware.

In this mode, the hypervisor emulates the full hardware, such as registers, timing, and hardware limitations. The guest operating system thinks it is interacting with real hardware. However, emulation is complex and inefficient.

#### Paravirtualization

In paravirtualization, the guest operating system knows that it is running inside a virtual machine, so it helps the hypervisor whenever possible. The advantage is the better performance of the virtual machine, mainly because the communication between hypervisor and guest can be shortened, which reduces overhead. With PowerKVM, all of the supported guests can run in paravirtualized mode.

Much of the paravirtualization optimization happens when the virtual machine operating system (OS) needs to do input and output (I/O) operations, which are processed by the hypervisor. One example is when the guest operating system needs to send a network packet outside of the server. When the guest OS sends the packet in full virtualization mode, it operates in the same way that it would when interacting with a physical NIC, using the same memory space, interruptions, and so on.

However, when the guest uses the paravirtualization approach, the guest operating system knows it is virtualized and knows that the guest I/O will arrive in a hypervisor (not on a physical hardware), and it cooperates with the hypervisor. This cooperation is what provides most of the performance benefits of paravirtualization.

In the context of KVM, this set of device drivers are called *Virtio* device drivers (see 1.3.11, “Virtio drivers” on page 21). There is a set of paravirtualized device drivers used initially on IBM PowerVM that is also supported on PowerKVM, including *ibmveth*, *ibmvscsi*, and others.

## Hardware-assisted virtualization

Hardware-assisted virtualization is a platform feature that enables the hypervisor to take advantage of the hardware when using guest virtualization. One of the main benefits is not changing the code of the guest images when running it, so the guest binary code can be run without any translation.

IBM Power Systems introduced virtualization assistance hardware with the POWER5 family of servers. At that time, Power Systems did much of the assistance by cooperating with the hypervisor for certain functions, such as fast page movement, micropartitioning, and Micro-Threading.

## 5.2 CPU overcommitment

CPU overcommitment allows an under-used CPU to be shared among other virtual machines. The CPU overcommit is usually enabled when the virtual machines are not expected to use all of the CPU resources at the same time. Therefore, when one virtual machine is not using its share of the CPU, another virtual machine can use it.

A CPU assigned to a virtual machine is called *virtual CPU* (vCPU). In an overcommitment scenario, the number of vCPUs is larger than the number of CPUs available.

For example, Figure 5-1 shows a hypervisor with four CPUs that is hosting two virtual machines (VMs) that are using three vCPUs each. This means that the guest operating system can use up to three CPUs if another VM is not using more than one CPU.

If the vCPU gets 100% used at a time, the hypervisor will multiplex the vCPU in the real CPU according to the hypervisor policies.

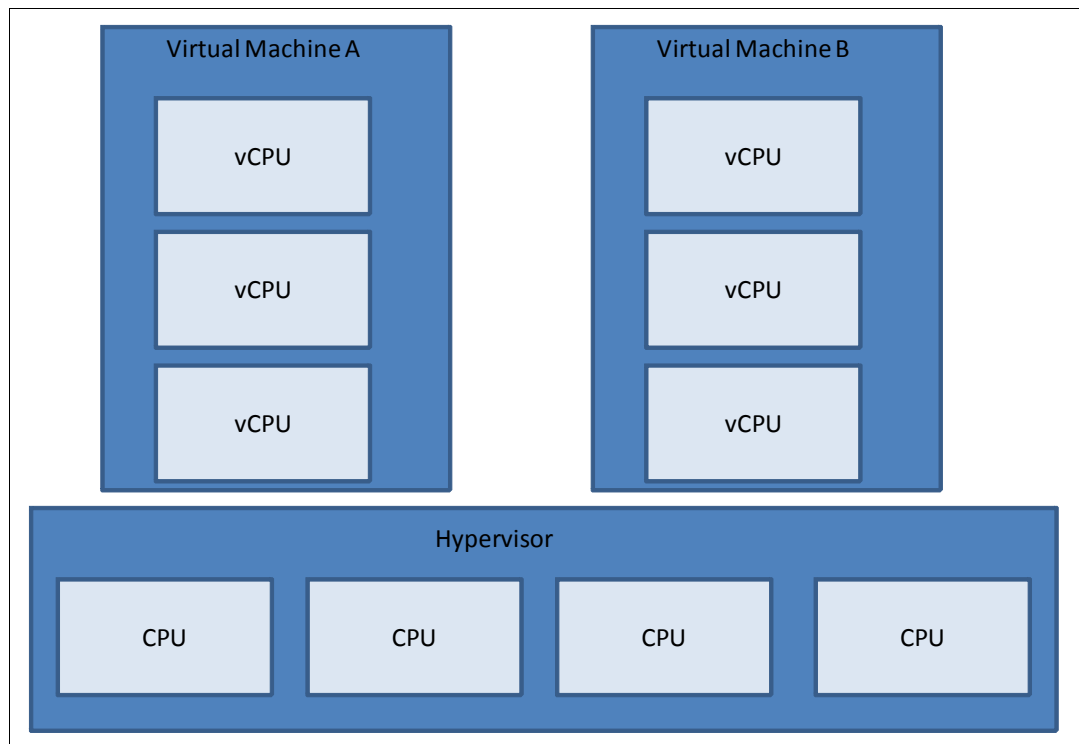


Figure 5-1 CPU overcommitment scenario

## 5.3 CPU configuration

There are many ways and techniques how to configure CPUs in a PowerKVM environment, which are discussed in the following sections.

### 5.3.1 CPU compatibility mode

It is possible to run a guest in compatibility mode with IBM POWER8, POWER7®, and POWER6® modes.

To enable POWER7 compatibility mode, add or edit the XML element in the domain element of the guest XML configuration file, as shown in Example 5-1.

*Example 5-1 Enable POWER7 compatibility mode*

---

```
<cpu mode='host-model'>
<model>power7</model>
</cpu>
```

---

**Note:** POWER7 compatibility mode is limited to up to four threads per core.

Example 5-2 shows how to verify the compatibility mode inside the guest. In this case, for POWER7.

*Example 5-2 Guest in POWER7 compatibility mode*

---

```
# cat /proc/cpuinfo
processor      : 0
cpu           : POWER7 (architected), altivec supported
clock        : 3026.000000MHz
revision     : 2.1 (pvr 004b 0201)

processor      : 1
cpu           : POWER7 (architected), altivec supported
clock        : 3026.000000MHz
revision     : 2.1 (pvr 004b 0201)

processor      : 2
cpu           : POWER7 (architected), altivec supported
clock        : 3026.000000MHz
revision     : 2.1 (pvr 004b 0201)

processor      : 3
cpu           : POWER7 (architected), altivec supported
clock        : 3026.000000MHz
revision     : 2.1 (pvr 004b 0201)

timebase      : 512000000
platform      : pSeries
model         : IBM pSeries (emulated by qemu)
machine       : CHRP IBM pSeries (emulated by qemu)
```

---

**Note:** The XML tag for the compatibility mode has been changed. In PowerKVM V2.1, it was `<cpu mode='custom'>`. In PowerKVM V3.1.0, the tag is `<cpu mode='host-model'>`. For the host migration from PowerKVM V2.1 to PowerKVM V3.1, scripts will take care of that change, as described in section 2.3, “Install over existing IBM PowerKVM and host migration” on page 52.

To enable POWER6 compatibility mode, add or edit the XML element shown in Example 5-3 on the domain element of the guest XML configuration file.

*Example 5-3 Enable POWER6 compatibility mode*

```
<cpu mode='host-model'>
<model>power6</model>
</cpu>
```

**Note:** POWER6 compatibility mode is limited to up to two threads per core.

### 5.3.2 Simultaneous multithreading

To run PowerKVM on Power Systems servers, the SMT option needs to be turned off in the hypervisor. The simultaneous multithreading (SMT) feature is visible only inside the guests, rather than on the hypervisor. In this scenario, a single core VM can use the SMT feature and have up to eight threads activated in the virtual machine.

To disable SMT on the hypervisor, run the following command:

```
ppc64_cpu --smt=off
```

PowerKVM disables SMT in the hypervisor during the boot. Each virtual machine that needs to use the SMT feature should enable it in the virtual machine configuration.

To check whether the SMT is disabled on the cores, run the `ppc64_cpu` command with the `--smt` or `--info` parameter. The `ppc64_cpu --info` command shows the output of the CPUs, marking the threads for each CPU that are enabled with an asterisk (\*) near the thread. Example 5-4 shows that in a six-core machine, only one thread per CPU is enabled.

*Example 5-4 SMT disabled on the hypervisor*

```
# ppc64_cpu --info
Core 0:  0*   1   2   3   4   5   6   7
Core 1:  8*   9  10  11  12  13  14  15
Core 2: 16*  17  18  19  20  21  22  23
Core 3: 24*  25  26  27  28  29  30  31
Core 4: 32*  33  34  35  36  37  38  39
Core 5: 40*  41  42  43  44  45  46  47
```

If you want to start the VM using SMT, it needs to specify that manually. For example, if you want to use only one core with SMT 8, the machine should be assigned with eight vCPUs, which will use just one core and eight threads, as covered in “SMT on the guests” on page 136.

To enable SMT support on a guest, the XML configuration file needs to set the number of threads per core. This number must be a power of 2, that is: 1, 2, 4, or 8. The number of vCPUs must also be the product of the number of threads per core and the number of cores.

Example 5-5 demonstrates how to set these numbers for four threads per core and two cores, resulting in eight vCPUs.

*Example 5-5 Setting the number of threads per core*

---

```
<vcpu placement='static'>8</vcpu>
<cpu>
<topology sockets='1' cores='2' threads='4' />
</cpu>
```

---

Example 5-6 shows the CPU information for the guest defined in Example 5-5. The guest is running with four threads per core and two cores. The example includes the information with SMT enabled and disabled.

*Example 5-6 CPU information about a guest with SMT*

---

```
# ppc64_cpu --smt
SMT=4

# cat /proc/cpuinfo
processor      : 0
cpu           : POWER8E (raw), altivec supported
clock         : 3026.000000MHz
revision      : 2.1 (pvr 004b 0201)

processor      : 1
cpu           : POWER8E (raw), altivec supported
clock         : 3026.000000MHz
revision      : 2.1 (pvr 004b 0201)

processor      : 2
cpu           : POWER8E (raw), altivec supported
clock         : 3026.000000MHz
revision      : 2.1 (pvr 004b 0201)

processor      : 3
cpu           : POWER8E (raw), altivec supported
clock         : 3026.000000MHz
revision      : 2.1 (pvr 004b 0201)

processor      : 4
cpu           : POWER8E (raw), altivec supported
clock         : 3026.000000MHz
revision      : 2.1 (pvr 004b 0201)

processor      : 5
cpu           : POWER8E (raw), altivec supported
clock         : 3026.000000MHz
revision      : 2.1 (pvr 004b 0201)

processor      : 6
cpu           : POWER8E (raw), altivec supported
clock         : 3026.000000MHz
revision      : 2.1 (pvr 004b 0201)

processor      : 7
```

```

cpu           : POWER8E (raw), altivec supported
clock        : 3026.000000MHz
revision     : 2.1 (pvr 004b 0201)

timebase     : 512000000
platform     : pSeries
model        : IBM pSeries (emulated by qemu)
machine      : CHRP IBM pSeries (emulated by qemu)

```

```

# 1scpu
Architecture:   ppc64le
Byte Order:     Little Endian
CPU(s):         8
On-line CPU(s) list: 0-7
Thread(s) per core: 4
Core(s) per socket: 2
Socket(s):      1
NUMA node(s):   1
Model:          IBM pSeries (emulated by qemu)
L1d cache:      64K
L1i cache:      32K
NUMA node0 CPU(s): 0-7

```

```
# ppc64_cpu --smt=off
```

```

# cat /proc/cpuinfo
processor      : 0
cpu           : POWER8E (raw), altivec supported
clock        : 3026.000000MHz
revision     : 2.1 (pvr 004b 0201)

processor      : 4
cpu           : POWER8E (raw), altivec supported
clock        : 3026.000000MHz
revision     : 2.1 (pvr 004b 0201)

timebase     : 512000000
platform     : pSeries
model        : IBM pSeries (emulated by qemu)
machine      : CHRP IBM pSeries (emulated by qemu)

```

```

# 1scpu
Architecture:   ppc64le
Byte Order:     Little Endian
CPU(s):         8
On-line CPU(s) list: 0,4
Off-line CPU(s) list: 1-3,5-7
Thread(s) per core: 1
Core(s) per socket: 2
Socket(s):      1
NUMA node(s):   1
Model:          IBM pSeries (emulated by qemu)
L1d cache:      64K
L1i cache:      32K
NUMA node0 CPU(s): 0,4

```

---

## SMT on the guests

To enable SMT on the guests, the virtual machine needs to be assigned with the number of threads that will run on the operating system. Keep the following formula in mind:

$$\text{vCPU} = \text{sockets} \times \text{cores} \times \text{threads}$$

Table 5-1 shows the relation between the number of vCPU in guests, according to the number of sockets, cores, and threads configured in the guest XML definition in libvirt.

Table 5-1 The relation between vCPU, cores, and threads on guest configuration

vCPU	Cores	SMT	Guest XML definition
32	4	8	<topology sockets='1' cores='4' threads='8' />
16	4	4	<topology sockets='1' cores='4' threads='4' />
8	4	2	<topology sockets='1' cores='4' threads='2' />
4	4	off	<topology sockets='1' cores='4' threads='1' />
16	2	8	<topology sockets='1' cores='2' threads='8' />
8	2	4	<topology sockets='1' cores='2' threads='4' />
4	2	2	<topology sockets='1' cores='2' threads='2' />
2	2	off	<topology sockets='1' cores='2' threads='1' />
8	1	8	<topology sockets='1' cores='1' threads='8' />
4	1	4	<topology sockets='1' cores='1' threads='4' />
2	1	2	<topology sockets='1' cores='1' threads='2' />
1	1	off	<topology sockets='1' cores='1' threads='1' />

## 5.3.3 Micro-Threading

*Micro-Threading* is an IBM POWER8 feature that enables each POWER8 core to be split into two or four subcores. Each subcore has also a limited number of threads, as listed in Table 5-2.

Table 5-2 Threads per subcore

Subcores per core	Threads per subcore
2	1, 2, 4
4	1, 2

This type of configuration provides performance advantages for some types of workloads.



Figure 5-2 shows the architecture of a POWER8 core using the Micro-Threading feature. In this scenario, the core is configured to have four subcores, and each subcore configured in two threads.

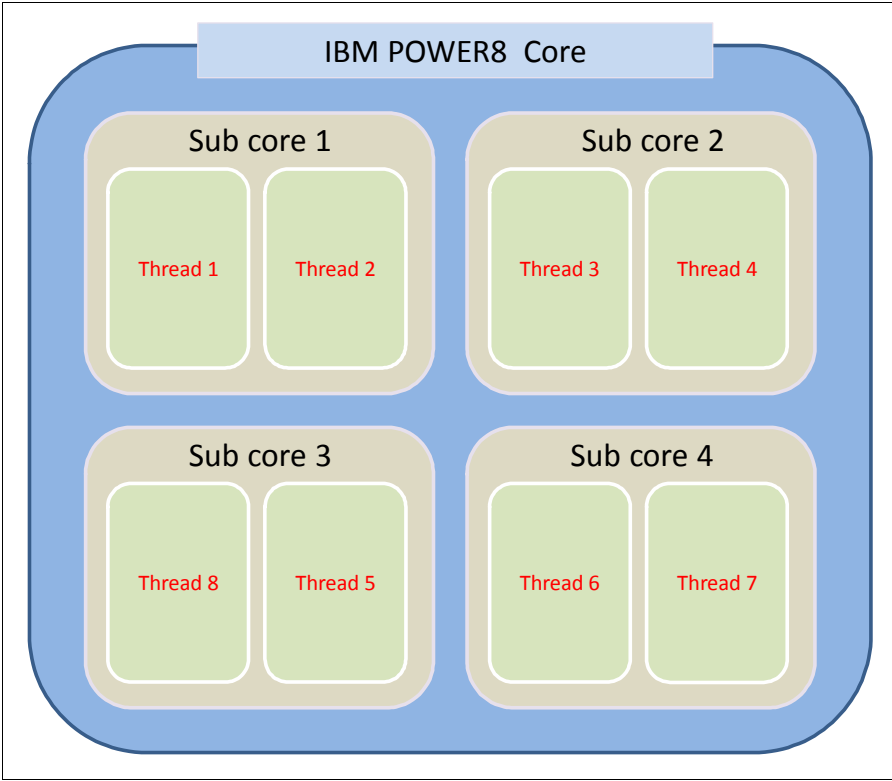


Figure 5-2 Example of a POWER8 core with four subcores and two threads each subcore

Another way to demonstrate how Micro-Threading works is defining a scenario where a user wants to start four virtual machines on a single core. You can start it without using Micro-Threading or with Micro-Threading enabled.

Figure 5-3 shows that four virtual machines are running in the same core, and each VM can access up to eight threads. The core switches among the four virtual machines, and each virtual machine runs only about one-fourth of the time. This indicates that the CPU is overcommitted.

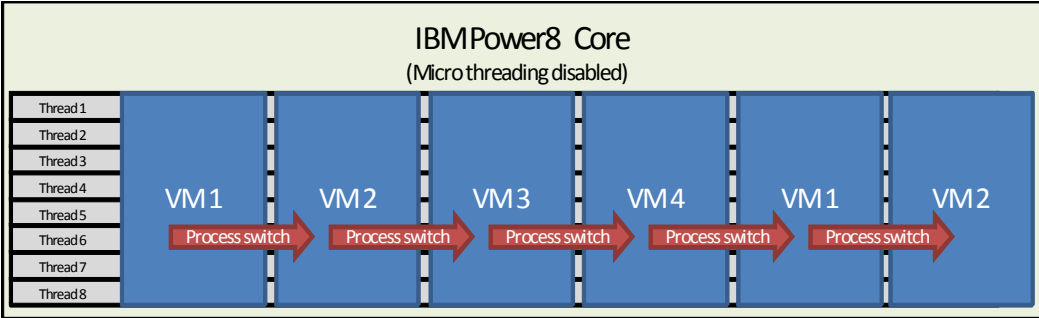


Figure 5-3 Four virtual machines running in a single core without Micro-Threading enabled

Figure 5-4 shows the same four virtual machines running on four different subcores in the same core. Each virtual machine can have up to two SMT threads. In this case, the guest is always running in the CPU.

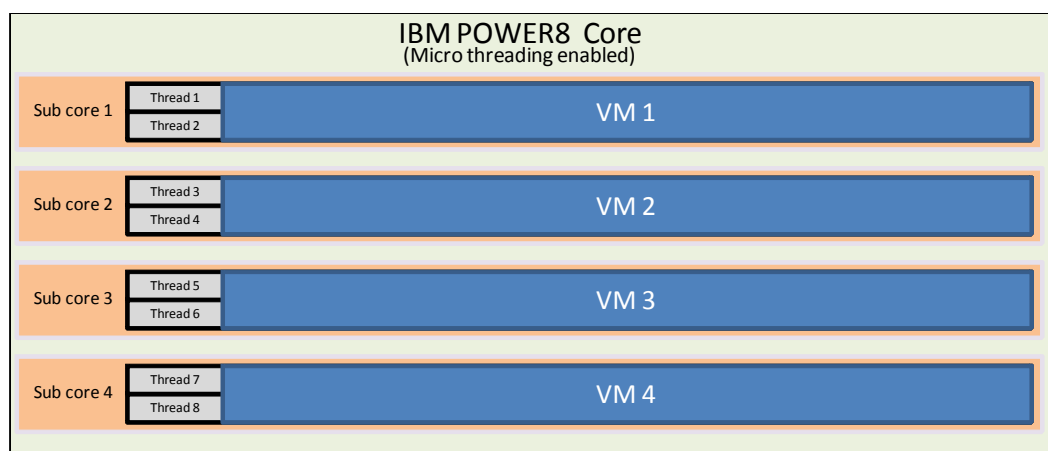


Figure 5-4 Four virtual machines running in a single core with Micro-Threading enabled

Micro-Threading benefits:

- ▶ Better CPU resources use
- ▶ More virtual machines per core

Micro-Threading limitations:

- ▶ SMT limited to 2 or 4 depending on the number of subcores
- ▶ Guests in single thread (SMT 1) mode cannot use the full core

## Dynamic Micro-Threading

PowerKVM V3.1 introduces *dynamic Micro-Threading*, which is enabled by default. Dynamic Micro-Threading allows virtual processors from several guests to run concurrently on the processor core. The processor core is split on guest entry and then made whole again on guest exit.

If the static Micro-Threading mode is set to anything other than whole core (in other words, set to 2 or 4 subcores) as described in “Enabling static Micro-Threading on the PowerKVM hypervisor” on page 139, dynamic Micro-Threading is disabled.

Along with dynamic Micro-Threading, PowerKVM V3.1 also implements a related feature called *subcore sharing*. Subcore sharing allows multiple virtual CPUs from the same guest to run concurrently on one subcore. Subcore sharing applies only to guests that are running in SMT 1 (whole core) mode and to virtual CPUs in the same guest. It applies in any Micro-Threading mode (static or dynamic).

Dynamic Micro-Threading can be also disabled or restricted to a mode that allows the core only to be dynamically split into two subcores or four subcores. This can be done by using the `dynamic_mt_modes` parameter.

Example 5-7 sets the parameter from the default 6 to 4, which means only splitting into four cores is allowed (not into two).

*Example 5-7 Only 4-way dynamic Micro-Threading*

```
# cat /sys/module/kvm_hv/parameters/dynamic_mt_modes
6

# echo 4 > /sys/module/kvm_hv/parameters/dynamic_mt_modes
```

Table 5-3 shows the supported values for `dynamic_mt_modes`.

*Table 5-3 Supported values for `dynamic_mt_modes`*

dynamic_mt_modes value	Result
0	Disables dynamic Micro-Threading
2	Allows 2-way Micro-Threading (but not 4-way Micro-Threading)
4	Allows 4-way Micro-Threading (but not 2-way Micro-Threading)
6	(= 4+2) ( <b>default</b> ): Allows both 2-way and 4-way Micro-Threading

**Note:** The documentation of dynamic Micro-Threading in the IBM Knowledge Center contains a table that shows the maximum number of virtual CPUs that can run on one core for the various Micro-Threading modes:

[http://www.ibm.com/support/knowledgecenter/SSZJY4\\_3.1.0/liabp/liabpdynamicsplit.htm](http://www.ibm.com/support/knowledgecenter/SSZJY4_3.1.0/liabp/liabpdynamicsplit.htm)

## Enabling static Micro-Threading on the PowerKVM hypervisor

To enable static Micro-Threading on the PowerKVM hypervisor, run the following procedures (the best way to do it is after a fresh reboot):

1. Ensure that all guests are *not* running.
2. Set the number of subcores to 1:  

```
# ppc64_cpu --subcores-per-core=1
```
3. Enable SMT on the host:  

```
# ppc64_cpu --smt=on
```
4. Set the number of subcores to 4:  

```
# ppc64_cpu --subcores-per-core=4
```
5. Turn the SMT off on the host:  

```
# ppc64_smt --smt=off
```

**Note:** To configure two subcores per core, specify `--subcores-per-core=2`.

To verify that the machine has Micro-Threading enabled, use the `ppc64_cpu` command and show the CPUs information with the `--info` parameters. Example 5-8 on page 140 shows the output of the `ppc64_cpu` command, displaying that the server has six cores and each core has four subcores.

Example 5-8 Checking if Micro-Threading is enabled

---

```
# ppc64_cpu --info
Core 0:
  Subcore 0: 0* 1
  Subcore 1: 2* 3
  Subcore 2: 4* 5
  Subcore 3: 6* 7
Core 1:
  Subcore 4: 8* 9
  Subcore 5: 10* 11
  Subcore 6: 12* 13
  Subcore 7: 14* 15
Core 2:
  Subcore 8: 16* 17
  Subcore 9: 18* 19
  Subcore 10: 20* 21
  Subcore 11: 22* 23
Core 3:
  Subcore 12: 24* 25
  Subcore 13: 26* 27
  Subcore 14: 28* 29
  Subcore 15: 30* 31
Core 4:
  Subcore 16: 32* 33
  Subcore 17: 34* 35
  Subcore 18: 36* 37
  Subcore 19: 38* 39
Core 5:
  Subcore 20: 40* 41
  Subcore 21: 42* 43
  Subcore 22: 44* 45
  Subcore 23: 46* 47
```

---

**Note:** If Micro-Threading is turned on with four subcores, and a guest is started that uses more than two threads, this results in the error **Cannot support more than 2 threads on PPC with KVM**. A four-thread configuration would be possible by activating Micro-Threading with only two subcores.

## Disabling static Micro-Threading

To disable the static Micro-Threading feature, follow these steps in the PowerKVM hypervisor:

1. Ensure that all guests are stopped.
2. Set the hypervisor cores back to full core mode:  
`ppc64_cpu --subcores-per-core=1`
3. Turn SMT on to “reset” the online thread topology:  
`ppc64_cpu --smt=on`
4. Turn the SMT off before starting the guests:  
`ppc64_cpu --smt=off`

To verify that the Micro-Threading feature is disabled, check with the `ppc64_cpu --info` command, as shown previously in Example 5-4 on page 133.

## 5.3.4 Configuring NUMA

*NUMA* stands for Non-Uniform Memory Access. It describes an environment where processors on different sockets, boards, or nodes have local memory that they can access directly, but also have access to memory at the other processors in the system. The far memory is also referred to as remote or distant memory. Local memory can be accessed faster as remote or distant memory. Therefore, it is best from a performance point of view if a guest only works with local memory.

Within PowerKVM, it is possible to define a NUMA environment on a guest. If that NUMA environment fits to the physical architecture of the system, that can result in better performance. To link the processors of a NUMA guest to the physical environment, CPU pinning can be used as described in 5.3.5, “CPU pinning” on page 142. Also, the memory can be linked to the physical environment of the system. This is done by restricting a guest to allocate memory from a set of NUMA nodes as described in 5.5.5, “Restrict NUMA memory allocation” on page 156.

A guest’s NUMA environment is defined in the CPU section of the domain in the XML file. Example 5-9 shows an environment of a system with two sockets and four cores in each socket. The guest should run in SMT8 mode. The NUMA section shows that the first 32 vCPUs (0 - 31) should be in NUMA cell 0 and the other 32 vCPUs (32 - 63) will be assigned to NUMA cell 1. The tag *current*=‘8’ in the vCPU section makes sure that the guest will start with only eight vCPUs, which is one core with eight threads. More CPUs can be later added using CPU Hotplug as described in 5.4, “CPU Hotplug” on page 145.

For the memory part of the guest, the XML file as shown in Example 5-9 defines that each cell should have 4 GB memory, equally spread over the two NUMA cells. The sum of the memory in the cells is also the maximum memory stated by the *memory* tag. If you try to set the maximum memory higher than the sum of the cells, PowerKVM automatically adjusts the maximum memory to the sum of the cells. Nevertheless, there is a possibility to have a higher maximum as the sum of memory in the cells by adding (virtual) dual inline memory modules (DIMMs) to the NUMA cells, as described in “Memory Hotplug in a NUMA configuration” on page 159.

*Example 5-9 Definition of a NUMA guest*

---

```
<memory unit='KiB'>8388338</memory>
  <currentMemory unit='KiB'>8388338</currentMemory>
  ...
<vcpu placement='static' current='8'>64</vcpu>
...
  <cpu>
    <topology sockets='2' cores='4' threads='8'/>
    <numa>
      <cell id='0' cpus='0-31' memory='4194304' unit='KiB'/>
      <cell id='1' cpus='32-63' memory='4194034' unit='KiB'/>
    </numa>
  </cpu>
```

---

To verify the result inside the guests, the **lscpu** and **numactl** commands can be used as shown in Example 5-10.

*Example 5-10 Verification of a NUMA configuration inside the guest*

---

```
# lscpu
Architecture:      ppc64le
Byte Order:        Little Endian
```

```

CPU(s):                8
On-line CPU(s) list:   0-7
Thread(s) per core:    8
Core(s) per socket:    1
Socket(s):              1
NUMA node(s):         2
Model:                 IBM pSeries (emulated by qemu)
L1d cache:              64K
L1i cache:              32K
NUMA node0 CPU(s):    0-7
NUMA node1 CPU(s):

```

```

# numactl -H
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7
node 0 size: 4096 MB
node 0 free: 2329 MB
node 1 cpus:
node 1 size: 4096 MB
node 1 free: 4055 MB
node distances:
node  0  1
  0:  10  10
  1:  10  10

```

---

### 5.3.5 CPU pinning

CPU *pinning* allows a guest virtual machine to be pinned to a given CPU or set of CPUs. It means that the hypervisor schedules only vCPUs in those CPUs that the guest is pinned to. By default, the guest can be scheduled on any CPU.

The advantage of pinning is that it can improve data locality. Two threads on the same core using the same data are able to share it on a local cache. The same thing happens for two cores on the same NUMA node.

Example 5-11 shows a configuration with four vCPUs without SMT turned on (SMT=1), where the four vCPUs are pinned to the first four cores in the first socket of the host.

*Example 5-11 CPU pinning without SMT*

---

```

<vcpu placement='static' cpuset='0,8,16,24'>4</vcpu>
<cpu>
<topology sockets='2' cores='6' threads='1' />
</cpu>

```

---

If the topology fits the system layout, for example within a Power System S812L with two physical sockets and six cores in each socket, this configuration makes sure that this guest only runs in the first socket of the system.

To verify if the pinning works correctly, the following commands can be used on the PowerKVM host as shown in Example 5-12.

*Example 5-12 Verification on CPU pinning*

---

```
# ppc64_cpu --info
Core 0: 0* 1 2 3 4 5 6 7 <-- Physical Socket 1
Core 1: 8* 9 10 11 12 13 14 15
Core 2: 16* 17 18 19 20 21 22 23
Core 3: 24* 25 26 27 28 29 30 31
Core 4: 32* 33 34 35 36 37 38 39
Core 5: 40* 41 42 43 44 45 46 47
Core 6: 48* 49 50 51 52 53 54 55 <-- Physical Socket 2
Core 7: 56* 57 58 59 60 61 62 63
Core 8: 64* 65 66 67 68 69 70 71
Core 9: 72* 73 74 75 76 77 78 79
Core 10: 80* 81 82 83 84 85 86 87
Core 11: 88* 89 90 91 92 93 94 95

# ps -ef | grep qemu | grep linux-guest
qemu      30179      1  5 17:44 ?          00:00:48 /usr/bin/qemu-system-ppc64 -name
linux-guest-1 -S -machine pseries-2.4,accel=kvm,usb=off -m
...

# taskset -cp 30179
pid 30179's current affinity list: 0,8,16,24
```

---

With SMT turned on in the guest, pinning CPUs works the same way, as SMT is not activated on the host. In an example with SMT 4, the first four guest vCPUs are mapped to threads 0, 1, 2, and 3 of the core 0 on the host. The second four guest vCPUs are mapped to threads 8, 9, 10, and 11 of the core 1 on the host, and so on. Example 5-13 shows the same configuration as in the previous example but with SMT 4.

*Example 5-13 CPU pinning with SMT*

---

```
<vcpu placement='static' cpuset='0,8,16,24'>16</vcpu>
<cpu>
<topology sockets='2' cores='6' threads='4'/>
</cpu>
```

---

**Note:** All threads of a core must be running on the same physical core. It is not supported to activate SMT on the PowerKVM host and pin single threads to different cores.

CPU pinning can be also used with subcores, which is explained in detail in 5.3.3, “Micro-Threading” on page 136. Also, in this case the pinning works in the same manner. In Example 5-14 on page 144, a guest using four subcores with two threads each is pinned to the first physical core.

*Example 5-14 CPU pinning with subcores*

---

```
# ppc64_cpu --info
Core 0:
  Subcore 0: 0* 1
  Subcore 1: 2* 3
  Subcore 2: 4* 5
  Subcore 3: 6* 7
Core 1:
  Subcore 4: 8* 9
  Subcore 5: 10* 11
  Subcore 6: 12* 13
  Subcore 7: 14* 15
Core 2:
  Subcore 8: 16* 17
  Subcore 9: 18* 19
  Subcore 10: 20* 21
  Subcore 11: 22* 23
...

<vcpu placement='static' cpuset='0,2,4,8'>48</vcpu>
<cpu>
<topology sockets='2' cores='6' threads='4'/>
</cpu>
```

---

### 5.3.6 CPU shares

In a kernel-based virtual machine (KVM), the virtual machines run as processes on the host. This means that they are scheduled to run on host CPUs just like any other process. The implication is that CPUs are shared by default. This CPU sharing allows CPU overcommitment, that is, creating more vCPUs than there are CPUs on the system.

The Linux scheduler spreads the vCPUs among the CPU cores. However, when there is overcommitment, multiple vCPUs can share a CPU core. To balance the amount of time that a virtual machine has compared to another virtual machine, you can configure shares.

Example 5-15 demonstrates how to configure the relative share time for a guest. By default, guests have a relative share time of 1024. Two guests with share time of 1024 shares the CPU for the same amount of time. If a third guest has a share time of 256, it runs a quarter of the time, relative to the other guests. A guest with a share time of 2048 runs twice the time compared to the other guests.

*Example 5-15 CPU shares*

---

```
<cputune>
<shares>256</shares>
</cputune>
```

---



This share time applies only when there is sharing either because of CPU pinning or because of CPU overcommitment. If vCPUs are idle or only a few vCPUs have been allocated, it is possible that a guest with a share time of 256 will be able to run on a CPU without sharing. If another guest needs to run on that same CPU, the configured share time will be in effect.

## 5.4 CPU Hotplug

Starting with PowerKVM V3.1, CPU Hotplug is supported. CPU Hotplug allows to add or remove CPUs in a running guest operating system. To support CPU Hotplug, the operating system needs the minimum required versions of the following packages.

*Table 5-4 Required packages to support CPU and memory Hotplug*

Package	Minimum required version
powerpc-utils	1.2.26
ppc64-diag	2.6.8
librtas	1.3.9

The addition or removal of CPUs is done on a per socket basis as defined in the CPU section in the guests XML file. A socket in that sense is not necessarily a physical socket of the Power System. It is just a virtual definition.

Before you start a hotplug operation, ensure that the `rtas_errd` daemon is running inside the guest:

```
# ps -ef | grep rtas
root      1367      1  0 09:22 ?          00:00:00 /usr/sbin/rtas_errd
```

The following examples were created on a Power System S812L with six cores on two sockets, giving a total of 12 cores in the system. The XML file of the guest system contains the following configuration as written in Example 5-16.

*Example 5-16 Base definition of sockets, cores, and threads for CPU Hotplug*

---

```
<vcpu placement='static' current='8'>96</vcpu>
...
<cpu>
<topology sockets='12' cores='1' threads='8' />
</cpu>
```

---

In Example 5-16, we defined a guest with 12 sockets, each with one core and eight threads, giving a total of 96 vCPUs. The guest will start with eight vCPUs, which is one socket with one CPU and eight threads, as defined with the attribute *current* in the *vcpu* section. From a CPU Hotplug perspective, the guest can be increased in steps of eight vCPUs up to 96 vCPUs (12 cores with eight threads).

The Hotplug task itself works in the same manner as PCI Hotplug works. An XML snippet is needed to define a sequence number for the additional socket. The snippet defining the first socket to be added begins with sequence number 0 as defined in Example 5-17.

*Example 5-17 XML snippet for Hotplugging a socket*

---

```
# cat cpu_hot_0.xml
<spapr-cpu-socket id="0">
<alias name="spaprcpusock0"/>
</spapr-cpu-socket>
```

---

**Note:** spapr-cpu-socket stands for Server IBM Power Architecture® Platform Reference CPU socket.

This snippet can be attached to the running guest with a **virsh attach-device** command as described in Example 5-18.

*Example 5-18 CPU Hotplug example*

---

```
[linux-guest]# lscpu
Architecture:      ppc64le
Byte Order:        Little Endian
CPU(s):           8
On-line CPU(s) list: 0-7
Thread(s) per core: 8
Core(s) per socket: 1
Socket(s):        1
NUMA node(s):      1
Model:             IBM pSeries (emulated by qemu)
L1d cache:         64K
L1i cache:         32K
NUMA node0 CPU(s): 0-7

[powerkvm-host]# virsh attach-device linux-guest cpu_hot_0.xml --live
Device attached successfully

[linux-guest]# lscpu
Architecture:      ppc64le
Byte Order:        Little Endian
CPU(s):           16
On-line CPU(s) list: 0-15
Thread(s) per core: 8
Core(s) per socket: 1
Socket(s):        2
NUMA node(s):      1
Model:             IBM pSeries (emulated by qemu)
L1d cache:         64K
L1i cache:         32K
NUMA node0 CPU(s): 0-15
```

---

**Note:** A persistent attachment of CPUs in the XML file by using the **--config** attribute is not supported.

In Example 5-18 on page 146, we added another socket with one core and eight threads. This can be repeated with a snippet containing the next available sequence number, for example 1. If the SMT type was changed in the meantime, CPU Hotplug is also possible. In that case, only the vCPUs matching the SMT mode will be online. The other vCPUs remain offline.

Example 5-19 continues Example 5-18 on page 146 by changing the SMT mode to 4 and adding another socket.

*Example 5-19 Change SMT mode and Hotplug another socket*

---

```
[linux-guest]# ppc64_cpu --smt
SMT=8
```

```
[linux-guest]# ppc64_cpu --smt=4
```

```
[linux-guest]# lscpu
Architecture:      ppc64le
Byte Order:        Little Endian
CPU(s):            16
On-line CPU(s) list: 0-3,8-11
Off-line CPU(s) list: 4-7,12-15
Thread(s) per core: 4
Core(s) per socket: 1
Socket(s):         2
NUMA node(s):      1
Model:             IBM pSeries (emulated by qemu)
L1d cache:         64K
L1i cache:         32K
NUMA node0 CPU(s): 0-3,8-11
```

```
[powerkvm-host]# cat cpu_hot_1.xml
<spapr-cpu-socket id="1">
<alias name="spaprcpusock1"/>
</spapr-cpu-socket>
```

```
[powerkvm-host]# virsh attach-device linux-guest cpu_hot_1.xml --live
Device attached successfully
```

```
[linux-guest]# lscpu
Architecture:      ppc64le
Byte Order:        Little Endian
CPU(s):            24
On-line CPU(s) list: 0-3,8-11,16-23
Off-line CPU(s) list: 4-7,12-15
Thread(s) per core: 5
Core(s) per socket: 1
Socket(s):         3
NUMA node(s):      1
Model:             IBM pSeries (emulated by qemu)
L1d cache:         64K
L1i cache:         32K
NUMA node0 CPU(s): 0-3,8-11,16-23
```

---

**Note:** It is not possible to attach two or more sockets with one snippet. However, this can be done by using several snippets in a loop.

Removing sockets using CPU Hotplug is also supported. To remove sockets, the same snippets are needed. The snippets must be applied using **virsh detach-device** in the opposite direction as the addition of the sockets. It is not possible to remove a lower sequence number before a higher sequence number. Example 5-20 shows the removal of one socket continuing the previous example.

*Example 5-20 Removal of one socket using CPU Hotplug*

---

```
[powerkvm-host]# virsh detach-device linux-guest cpu_hot_0.xml --live
error: Failed to detach device from cpu_hot_0.xml
error: unsupported configuration: Non-contiguous socket index '0' not allowed.
Expecting : 1
```

```
[powerkvm-host]# virsh detach-device linux-guest cpu_hot_1.xml --live
Device detached successfully
```

```
[linux-guest]# lscpu
Architecture:      ppc64le
Byte Order:        Little Endian
CPU(s):            16
On-line CPU(s) list: 0-3,8-11
Off-line CPU(s) list: 4-7,12-15
Thread(s) per core: 4
Core(s) per socket: 1
Socket(s):         2
NUMA node(s):      1
Model:             IBM pSeries (emulated by qemu)
L1d cache:         64K
L1i cache:         32K
NUMA node0 CPU(s): 0-3,8-11
```

---

### 5.4.1 CPU Hotplug with a NUMA configuration

For a NUMA guest, the NUMA node to which a CPU will be hotplugged, is determined by the NUMA topology defined in the guest XML. Example 5-21 shows a two-NUMA cells configuration with two sockets each and how the sockets will be populated using CPU Hotplug, according to the NUMA definition in the XML file.

*Example 5-21 CPU Hotplug with a NUMA configuration*

---

```
[powerkvm-host]# virsh edit linux-guest
...
<vcpu placement='static' current='8'>32</vcpu>
...
  <cpu>
    <topology sockets='4' cores='1' threads='8'/>
    <numa>
      <cell id='0' cpus='0-15,16-23' memory='2097152' unit='KiB'/>
      <cell id='1' cpus='8-15,24-31' memory='2097152' unit='KiB'/>
    </numa>
  </cpu>
```

...

```
[linux-guest]# lscpu
Architecture:      ppc64le
Byte Order:        Little Endian
CPU(s):           8
On-line CPU(s) list: 0-7
Thread(s) per core: 8
Core(s) per socket: 1
Socket(s):          1
NUMA node(s):      2
Model:             IBM pSeries (emulated by qemu)
L1d cache:          64K
L1i cache:          32K
NUMA node0 CPU(s): 0-7
NUMA node1 CPU(s):
```

```
[powerkvm-host]# virsh attach-device linux-guest cpu_hot_0.xml --live
Device attached successfully
```

```
[linux-guest]# lscpu
Architecture:      ppc64le
Byte Order:        Little Endian
CPU(s):           16
On-line CPU(s) list: 0-15
Thread(s) per core: 8
Core(s) per socket: 1
Socket(s):          2
NUMA node(s):      2
Model:             IBM pSeries (emulated by qemu)
L1d cache:          64K
L1i cache:          32K
NUMA node0 CPU(s): 0-7
NUMA node1 CPU(s): 8-15
```

```
[powerkvm-host]# virsh attach-device linux-guest cpu_hot_1.xml --live
Device attached successfully
```

```
[linux-guest]# lscpu
Architecture:      ppc64le
Byte Order:        Little Endian
CPU(s):           24
On-line CPU(s) list: 0-23
Thread(s) per core: 8
Core(s) per socket: 1
Socket(s):          3
NUMA node(s):      2
Model:             IBM pSeries (emulated by qemu)
L1d cache:          64K
L1i cache:          32K
NUMA node0 CPU(s): 0-7,16-23
NUMA node1 CPU(s): 8-15
```

```
[powerkvm-host]# virsh attach-device linux-guest cpu_hot_2.xml --live
Device attached successfully
```

```
[linux-guest]# lscpu
Architecture:      ppc64le
Byte Order:        Little Endian
CPU(s):            32
On-line CPU(s) list: 0-31
Thread(s) per core: 8
Core(s) per socket: 1
Socket(s):          4
NUMA node(s):      2
Model:             IBM pSeries (emulated by qemu)
L1d cache:         64K
L1i cache:         32K
NUMA node0 CPU(s): 0-7,16-23
NUMA node1 CPU(s): 8-15,24-31
```

---

## 5.4.2 Considerations for CPU Hotplug

There are some considerations when using CPU Hotplug.

### **No removal of sockets that were present at the time of starting the guest**

Only added Hotplug sockets can be removed by using a Hotplug action. If for example the guest was started with two sockets (as defined in the XML definition), and one socket should be removed by using `virsh detach-device`, this results in an error.

### **No CPU Hotplug with unfilled sockets**

CPU Hotplug is not possible in a configuration where a socket was not completely used at starting time. The following configuration in Example 5-22 does not support CPU Hotplug. The guest starts with only four vCPUs, which means that only one of the two cores of the socket is used.

*Example 5-22 Unsupported configuration for CPU Hotplug*

---

```
<vcpu placement='static' current="4">32</vcpu>
<cpu>
  <topology sockets='4' cores='2' threads='4' />
</cpu>
```

---

## 5.5 Memory

With virtualization, the memory is basically static, which means it is not virtualized like the CPU, and a block of memory is mapped directly to a single (and just one) virtual machine.

Because each virtual machine is also a hypervisor Linux thread, the memory can be overcommitted.

This section covers methods to improve the performance of PowerKVM memory management. These methods involve resizing the guest memory dynamically and merging identical guests pages on the hypervisor.

## 5.5.1 Memory allocation

Guest memory is allocated by the host according to the guest configuration. It is possible to set a maximum amount of memory and a current amount. The guest will have the maximum amount of memory available, but it can choose to use only the current amount and release the remaining amount to the host. See 5.5.2, “Memory ballooning” on page 151.

Example 5-23 shows the configuration for the maximum amount of memory allocated to the guest on the *memory* element and the current amount of memory on the *currentMemory* element. Since PowerKVM V3.1, it is possible to also increase the memory across the maximum amount by using memory hotplug as described in 5.6, “Memory Hotplug” on page 157.

*Example 5-23 Memory allocation*

---

```
<memory unit='KiB'>4194304</memory>
<currentMemory unit='KiB'>2097152</memory>
```

---

**Note:** On the guest, you might notice that there is a total amount of memory that is less than what is set as the current amount. This might happen because the guest subcore has reserved an amount of memory for some reason. One example is the **crashkernel** command, which is used for a kernel dump.

## 5.5.2 Memory ballooning

*Memory ballooning* is a technique that allows the guest memory to be increased or decreased cooperatively, depending on the amount of free memory available on the guests and hypervisor.

When memory ballooning is enabled on the guest, the hypervisor can remove and add memory to the guest dynamically.

This technique can be used if the memory should be overcommitted, which means assigning the guests in sum more memory that the system provides. In case a guest needs more memory and another guest needs less memory at the same time, the memory is used more efficiently. But if all guests need their assigned overcommitted memory, this can cause a bad performance because in that case the host starts to swap pages to disk.

### How to enable and manage memory ballooning on PowerKVM

Memory ballooning is enabled by default using the *virtio memballoon* model as shown in Example 5-24. Only if you want to disable ballooning, change the model to *none*.

*Example 5-24 Enable memory balloon on the guest*

---

```
<devices>
..
<memballoon model='virtio'>
..
</devices>
```

---

When a guest is configured to support ballooning, the memory can be added and removed from the virtual machine using the **virsh setmem linux-guest** command. The total memory allocated to the virtual machine can be seen with the **virsh dommemstat** command.

Example 5-25 shows a virtual machine initially with 2 GB memory. After the **virsh setmem linux-guest 1048576 --config --live** command, the memory assigned to that partition goes to 1 GB. The **--live** flag changes the amount of memory in the running guest and the **--config** changes the currentMemory unit tag in the XML file. The two flags can be used individually or together.

*Example 5-25 Decreasing the virtual machine memory to 1 GB*

---

```
# virsh dommemstat linux-guest
actual 2097152
swap_in 46312795184
rss 1954752

# virsh setmem linux-guest 1048576 --config --live

# virsh dommemstat linux-guest
actual 1048576
swap_in 46312795184
rss 1955200
```

---

**Note:** If the virtual machine or the guest operating system is not configured properly to support virtio ballooning, the following message displays on the hypervisor:

Error: Requested operation is not valid: Unable to change memory of active domain without the balloon device and guest OS balloon drive.

## Monitoring

To check whether the memory ballooning is working on the guest, you can check with the QEMU monitor that is running the command, as shown in Example 5-26. If the balloon is not available in the virtual machine, the output is “Device balloon has not been activated.”

*Example 5-26 Output of memory available on the balloon*

---

```
# virsh qemu-monitor-command --domain linux-guest --hmp 'info balloon'
ballon: actual=3559
```

---

To change the amount of memory in the guest, the **'balloon <memory in MB>'** command is used, as in Example 5-27, that changes the memory from 3559 MB to 1024 MB. After this command, only 1024 MB of memory is available to the guest.

*Example 5-27 Changing the memory allocated to the virtual machine*

---

```
(qemu) virsh qemu-monitor-command --domain linux-guest --hmp 'info balloon'
ballon: actual=3559
(qemu) virsh qemu-monitor-command --domain linux-guest --hmp 'balloon 1024'
(qemu) virsh qemu-monitor-command --domain linux-guest --hmp 'info balloon'
ballon: actual=1024
```

---

**Note:** Most of the operating systems have virtio-balloon embedded into the kernel. If you are using an operating system that does not have the virtio-balloon device driver in the kernel, you need to install it manually.



### 5.5.3 Kernel SamePage Merging

Kernel SamePage Merging (KSM) is a KVM technology that merges blocks of memory pages with the same content to reduce the memory use in the hypervisor.

KSM technology can detect that two virtual machines have identical memory pages. In that case, it merges both pages in the same physical memory page, which reduces that amount of memory use. To do so, a certain number of CPU cycles is used to scan and spot these pages.

For example, Figure 5-5 shows that all three virtual machines have pages that contain the same content. In this case, when KSM is enabled, all four pages that contain the same content will use only one physical memory block.

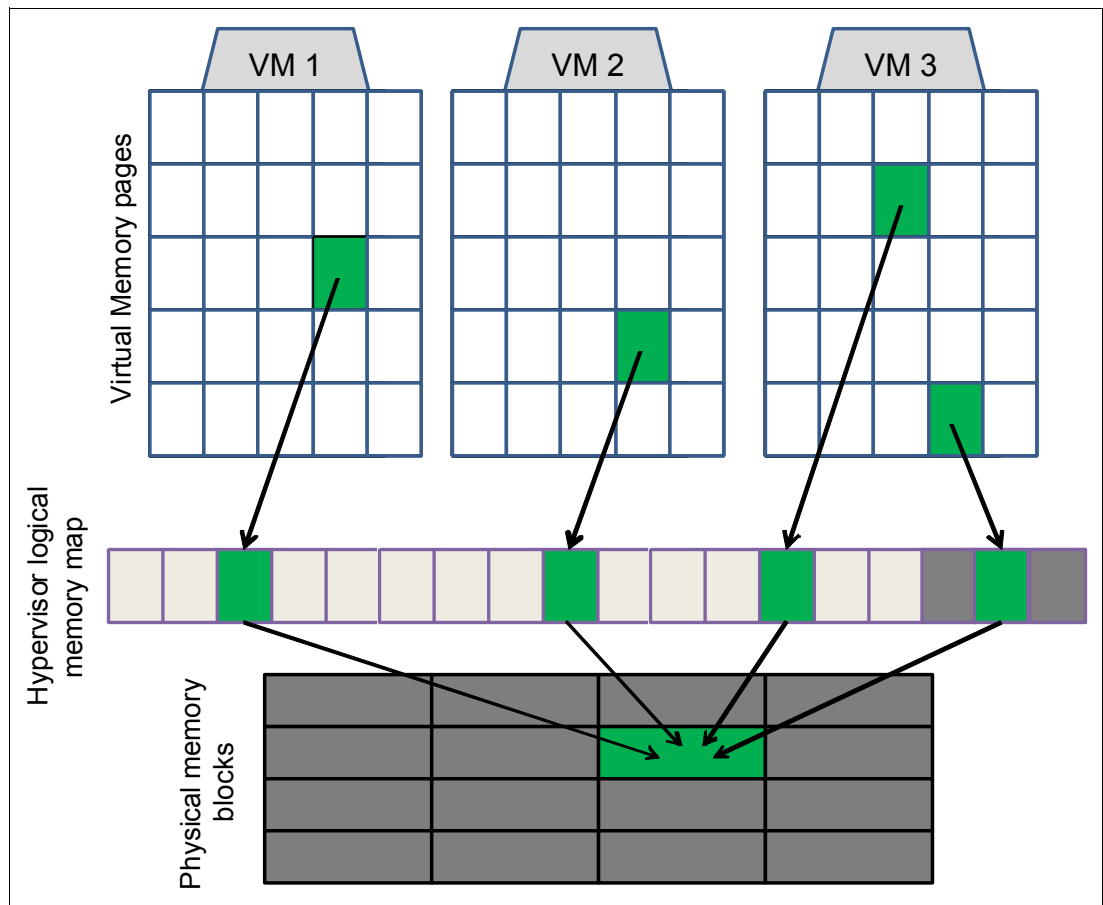


Figure 5-5 KSM mapping when VM uses the same page

There is a similar feature found in the PowerVM hypervisor, called *Active Memory Deduplication*. For more information about this feature, see “*Power Systems Memory Deduplication*, REDP-4827.”

#### How to enable Kernel SamePage Merging on PowerKVM

KSM is supported in PowerKVM server virtualization, but it is not enabled automatically.

To verify whether KSM is running and to enable and disable it, you need to interact with the `/sys/kernel/mm/ksm/run` file.

**Important:** The *ksmtuned* daemon must be running to run KSM. PowerKVM already has this daemon running automatically, so you do not need to turn it on. To verify that the daemon is running, check Example 5-28.

*Example 5-28 Verify that the ksmtuned daemon is running*

---

```
# systemctl status ksmtuned
ksmtuned.service - Kernel Samepage Merging (KSM) Tuning Daemon
  Loaded: loaded (/usr/lib/systemd/system/ksmtuned.service; enabled)
  Active: active (running) since Sat 2014-05-10 10:55:52 EDT; 2 days ago
  Main PID: 18420 (ksmtuned)
  CGroup: name=systemd:/system/ksmtuned.service
          17510 sleep 60
          18420 /bin/bash /usr/sbin/ksmtuned
```

---

Example 5-29 shows that KSM is disabled and how to enable it.

*Example 5-29 Enable KSM in PowerKVM*

---

```
# cat /sys/kernel/mm/ksm/run
0
# echo 1 > /sys/kernel/mm/ksm/run
# cat /sys/kernel/mm/ksm/run
1
```

---

## Monitoring KSM

To monitor the pages being merged by KSM, check the `/sys/kernel/mm/ksm` files. The subsections that follow explain some of the status files.

### ***Pages shared***

The `/sys/kernel/mm/ksm/pages_shared` file shows how many merged pages exist in the system. Example 5-30 shows that 2976 pages are shared by two or more virtual machines in the system.

*Example 5-30 Number of pages shared in the hypervisor*

---

```
# cat /sys/kernel/mm/ksm/page_shared
2976
```

---

### ***Pages sharing***

The `/sys/kernel/mm/ksm/pages_sharing` file shows how many pages on the virtual machines are using a page that is shared and merged in the hypervisor. Example 5-31 shows the number of pages in the virtual machines that are linked to a shared page in the hypervisor.

*Example 5-31 Number of pages that are linked to a shared page*

---

```
# cat /sys/kernel/mm/ksm/page_sharing
6824
```

---

Looking at both of the previous examples, you see that 6824 virtual pages are using 2976 physical pages, which means that 3848 pages are saved. Considering 64 KB pages, this means that approximately 246 MB of memory was saved by using this feature.

There are some other monitoring options for KSM, as shown in Table 5-5.

<b>/sys/kernel/mm/ksm options</b>	<b>Description</b>
pages_unshared	How many pages are candidates to be shared but are not shared at the moment
pages_volatile	The number of pages that are candidates to be shared but are being changed so frequently that they will not be merged
full_scans	How many times the KSM scanned the pages looking for duplicated content
merge_across_nodes	Option to enable merges across NUMA nodes (disable it for better performance)
pages_to_scan	How many pages the KSM algorithm scans per turn before sleeping
sleep_miliseconds	How many milliseconds <b>ksmd</b> should sleep before the next scan

Table 5-5 KSM options

## 5.5.4 Huge pages

*Huge pages* is a Linux feature that uses the processor capability to use multiple page sizes. POWER processors support multiple page sizes since POWER5. Some workloads benefit from using a larger page size. IBM Power Systems that run Linux can use 16 MiB page sizes.

On IBM PowerKVM, a guest must have its memory backed by huge pages for the guest to be able to use it. You need to enable huge pages on the host and configure the guest to use huge pages before you start it.

Example 5-32 demonstrates how to enable huge pages on the host. Run the command on a host shell. The number of pages to use depends on the total amount of memory for guests that are backed by huge pages. In this example, 4 GB of memory is reserved for huge pages (256 pages with 16384 KB each).

Example 5-32 Setting huge pages on the host

```
# echo 256 > /proc/sys/vm/nr_hugepages
# grep -i hugepage /proc/meminfo
HugePages_Total:    256
HugePages_Free:     256
HugePages_Rsvd:      0
HugePages_Surp:      0
Hugepagesize:       16384 kB
```

Example 5-33 shows an excerpt from an XML configuration file for a guest, demonstrating how to enable huge pages. The **memoryBacking** element must be inside the domain element of the XML configuration file.

Example 5-33 Enabling huge pages on a guest

```
<memoryBacking>
<hugepages/>
</memoryBacking>
```

If there are not enough huge pages to back your guest memory, you will see the error in Example 5-34. Try increasing the number of huge pages on the host.

*Example 5-34 Error starting a guest with huge pages*

---

```
# virsh start linux-guest
error: Failed to start domain linux-guest
error: internal error: early end of file from monitor: possible problem:
2015-11-04T17:46:01.720148Z qemu-system-ppc64: unable to map backing store for
hugepages: Cannot allocate memory
```

---

## 5.5.5 Restrict NUMA memory allocation

It is possible to restrict a guest to allocate memory from a set of NUMA nodes. If the guest vCPUs are also pinned to a set of cores on that same set of NUMA nodes, memory access will be local, which improves memory access performance.

Example 5-35 presents the output of a command on the PowerKVM host that shows how many pages have been allocated on every node **before** restricting the guest to only one NUMA node.

*Example 5-35 Memory allocation to NUMA nodes before restricting it to one node*

---

```
# cat
/sys/fs/cgroup/memory/machine.slice/machine-qemu\x2dlinux-guest\x2dl.scope/memory.numa_stat
total=27375 N0=23449 N1=3926
file=0 N0=0 N1=0
anon=27375 N0=23449 N1=3926
unevictable=0 N0=0 N1=0
hierarchical_total=27375 N0=23449 N1=3926
hierarchical_file=0 N0=0 N1=0
hierarchical_anon=27375 N0=23449 N1=3926
hierarchical_unevictable=0 N0=0 N1=0
```

---

The output shows that most of the memory is assigned to NUMA node 0 (N0) but some memory to NUMA node 1 (N1).

**Note:** The path in the command contains the name of the guest (in Example 5-35 linux-guest) and is only available when the guest is running.

Example 5-36 presents a possible configuration to restrict a guest to NUMA node 0.

*Example 5-36 NUMA node set*

---

```
<numatune>
<memory nodeset='0' />
</numatune>
```

---

**Note:** To find out how many nodes a system contains, use the `numactl -H` command. An example output is contained in Example 5-42 on page 159.

After restarting the guest and if the system has enough free memory on NUMA node 0, the command lists that all memory now fits into NUMA node 0 as shown in Example 5-37 on page 157.

*Example 5-37 Memory allocation to NUMA nodes after restricting it to one node*

---

```
# cat
/sys/fs/cgroup/memory/machine.slice/machine-qemu\x2dlinux-guest\x2d1.scope/memory.numa_stat
total=24751 N0=24751 N1=0
file=0 N0=0 N1=0
anon=24751 N0=24751 N1=0
unevictable=0 N0=0 N1=0
hierarchical_total=24751 N0=24751 N1=0
hierarchical_file=0 N0=0 N1=0
hierarchical_anon=24751 N0=24751 N1=0
hierarchical_unevictable=0 N0=0 N1=0
```

---

**Note:** The number of memory pages shown here is used pages by the guest. Therefore, the number changes over time.

## 5.6 Memory Hotplug

Memory Hotplug was introduced in PowerKVM V3.1 and allows to increase the memory over the maximum amount of memory that is defined by the `memory` attribute in the XML file. Memory Hotplug uses up to 32 (virtual) hotpluggable DIMM modules that can be added to a domain. The hotplugged DIMM modules can be different in size and are not limited to a maximum amount. Only the guest definition limits the maximum amount of a DIMM that can be added.

Only adding of memory is supported. It is not possible to remove DIMMs that were added using memory hotplug. Memory Hotplug assigns contiguous chunks of memory to the guest. By adding memory using memory ballooning this is not necessarily the case, which can result in memory fragmentation. Although it is possible to also reduce the memory with memory ballooning if the guest supports it, as described in 5.5.2, “Memory ballooning” on page 151.

Before using memory hotplug, ensure that the guest operating system has the required packages installed as listed in Table 5-4 on page 145.

Like CPU Hotplug, a memory DIMM can be added by using an XML snippet that defines the size of the DIMM that should be added. Example 5-38 shows a snippet for a DIMM of 4 GB.

*Example 5-38 XML snippet for a DIMM with 4 GB*

---

```
<memory model='dimm'>
<target>
<size unit='KiB'>4194304</size>
</target>
</memory>
```

---

**Note:** In comparison to CPU Hotplug, there is no sequence number needed. That means a snippet can be used several times for one running guest.

To attach a memory DIMM to a running domain, use the **virsh attach-device** **<snipped.xml>** **--live** command for adding CPU or other devices. In Example 5-39 on page 158, we show how to increase the memory of a guest with a maximum of 4 GB of memory first by 2 GB and then by another 4 GB giving 10 GB in total. After that, we reduce the memory by using ballooning back to 4 GB.

Example 5-39 Example of how to increase the memory by using memory hotplug

---

```
[powerkvm-host]# virsh dumpxml linux-guest
...
<maxMemory slots='32' unit='KiB'>67108864</maxMemory>
  <memory unit='KiB'>4194304</memory>
  <currentMemory unit='KiB'>4194304</currentMemory>
...

[linux-guest]# free -m
              total          used          free      shared  buff/cache   available
Mem:           3558           587          2378           19         592         2812
Swap:           1023              0          1023

[powerkvm-host]# cat mem_hot_2G.xml
<memory model='dimm'>
<target>
<size unit='KiB'>2097152</size>
</target>
</memory>

[powerkvm-host]# virsh attach-device linux-guest mem_hot_2G.xml --live
Device attached successfully

[linux-guest]# free -m
              total          used          free      shared  buff/cache   available
Mem:           5606           615          4367           19         623         4817
Swap:           1023              0          1023

[powerkvm-host]# cat mem_hot_2G.xml
<memory model='dimm'>
<target>
<size unit='KiB'>4194304</size>
</target>
</memory>

[powerkvm-host]# virsh attach-device linux-guest mem_hot_4G.xml --live
Device attached successfully

[linux-guest]# free -m
              total          used          free      shared  buff/cache   available
Mem:           9702           635          8442           19         625         8883
Swap:           1023              0          1023

[powerkvm-host]# virsh dommemstat linux-guest
actual 10485760
swap_in 0
rss 1879744

[powerkvm-host]# virsh setmem linux-guest 4194304 --live

[linux-guest]# free -m
              total          used          free      shared  buff/cache   available
Mem:           3558           618          2315           19         625         2756
Swap:           1023              0          1023
```

---

In Example 5-39 on page 158, we reduced the amount of memory back to its original value, but remember that nevertheless two DIMMs were added to the running guest and are still added. That means only 30 DIMMs (out of 32) are left for being hotplugged.

**Remember:** It is not possible to remove the added DIMMs by using the memory hotplug function.

Memory DIMMs can be also added persistently to the configuration of the guest by adding `--config` to the attach command as shown in Example 5-40. The DIMMs are added into the devices section of the guest XML.

*Example 5-40 Persistent attachment of DIMMs*

---

```
# virsh attach-device linux-guest mem_hot_1G.xml --live --config

# virsh edit linux-guest
...
<devices>
...
  <memory model='dimm'>
    <target>
      <size unit='KiB'>1048576</size>
      <node>0</node>
    </target>
  </memory>
...
</devices>
```

---

This section describes additional options and possibilities used with memory hotplug.

## Memory Hotplug in a NUMA configuration

Memory Hotplug can be also used within a NUMA configuration. In this case, the NUMA node is specified in the XML snippet. Example 5-41 shows a snippet defining that the memory DIMM should be added to node 1.

*Example 5-41 Memory Hotplug snippet in a NUMA environment*

---

```
<memory model='dimm'>
<target>
<size unit='KiB'>1048576</size>
<node>1</node>
</target>
</memory>
```

---

Example 5-42 shows how to attach 1 GB of memory to just NUMA node 1 by using the snippet as shown in Example 5-41.

*Example 5-42 Memory Hotplug within a NUMA configuration*

---

```
[linux-guest]# numactl -H
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7
node 0 size: 2048 MB
node 0 free: 1074 MB
node 1 cpus: 8 9 10 11 12 13 14 15
```

---

```
node 1 size: 2048 MB
```

```
node 1 free: 1152 MB
```

```
node distances:
```

```
node 0 1
```

```
0: 10 40
```

```
1: 40 10
```

```
[powerkvm-host]# virsh attach-device linux-guest mem_hot_1G_numa1.xml --live  
Device attached successfully
```

```
[linux-guest]# numactl -H
```

```
available: 2 nodes (0-1)
```

```
node 0 cpus: 0 1 2 3 4 5 6 7
```

```
node 0 size: 2048 MB
```

```
node 0 free: 1036 MB
```

```
node 1 cpus: 8 9 10 11 12 13 14 15
```

```
node 1 size: 3072 MB
```

```
node 1 free: 2166 MB
```

```
node distances:
```

```
node 0 1
```

```
0: 10 40
```

```
1: 40 10
```

---

Also in a NUMA environment, the DIMMs can be added persistently by adding **--config** to the **virsh attach-device** command. As a result, the DIMMs are added including the correct cell (node) definition for the DIMMs as shown in Example 5-43. The example also shows that in this case, the maximum memory of the guest is higher than the sum of memory defined in the NUMA section of the XML file.

---

*Example 5-43 Persistent Hotplug Memory DIMMs*

---

```
# virsh attach-device linux-guest mem_hot_1G_numa1.xml --live --config
```

```
# virsh edit linux-guest
```

```
...
```

```
<memory unit='KiB'>9436914</memory>
```

```
<cpu>
```

```
<topology sockets='2' cores='4' threads='8'>
```

```
<numa>
```

```
<cell id='0' cpus='0-31' memory='4194304' unit='KiB'>
```

```
<cell id='1' cpus='32-63' memory='4194034' unit='KiB'>
```

```
</numa>
```

```
</cpu>
```

```
...
```

```
<device>
```

```
<memory model='dimm'>
```

```
<target>
```

```
<size unit='KiB'>1048576</size>
```

```
<node>1</node>
```

```
</target>
```

```
</memory>
```

```
</device>
```

---



**Huge pages support**

Guests that use huge pages are also supported by memory hotplug. If enough huge pages are available, these can be added to a guest using the same methodology as described in this chapter. For more information about huge pages, see 5.5.4, “Huge pages” on page 155.

**Live migration support**

Guests with virtual DIMMs added using memory hotplug can also be migrated to a different host.





## I/O virtualization

*I/O virtualization* is the process of allowing a single I/O adapter to be used by many virtual machines. It is very common for network, storage, and USB.

This chapter covers the following topics:

- ▶ Hypervisor virtualization
- ▶ Hardware virtualization
- ▶ Device assignment without virtualization (pass-through)
- ▶ Fibre Channel virtualization (NPIV)
- ▶ Multipath disk usage

## 6.1 Types of virtualization

With hypervisor virtualization, the I/O device is virtualized by the hypervisor. In this model, the hypervisor needs to multiplex the virtual devices and control the I/O on the real devices.

### 6.1.1 PowerKVM supported devices

As part of I/O virtualization, IBM PowerKVM supports the following device modes:

- ▶ Storage
  - virtio-scsi
  - virtio
  - spapr-vscsi
- ▶ Network
  - virtio
  - e1000 (Intel)
  - RTL (Realtek)
  - spapr-vlan
- ▶ Graphic models
  - VGA

**Note:** The SPICE graphical model is not supported for IBM PowerKVM V3.1.

### 6.1.2 PCI I/O pass-through

I/O pass-through is not a method of I/O virtualization. It consists of giving a guest direct access to an I/O device. The adapter is assigned to a virtual machine, and the virtual machine can use this I/O exclusively.

For more information about I/O pass-through, see 6.4, “I/O pass-through” on page 170.

## 6.2 Network virtualization

A virtual machine frequently needs to have network connectivity to access external servers and other virtual machines. The network is an important area in virtualization. Currently, there are different ways to allow a virtual machine to access the external network, and those methods are explained briefly in this chapter.

A network card can be assigned directly to the VM by using the I/O pass-through method.

These are the current network virtualization methods that are available:

- ▶ User mode networking
- ▶ Network address translation (NAT) networking
- ▶ Bridged networking
- ▶ PCI pass-through
- ▶ Open vSwitch

See 3.4, “Network” on page 75 for how to create NAT and bridged networks by using Kimchi.

## 6.2.1 User mode networking

In this scenario, the traffic is sent through QEMU by using a user space socket that drives the packet to the whole network. This mode also uses the concept of NAT to translate packets from one VM to the external network. The big advantage of user mode networking is that the users on the system do not need to have privileged permissions to enable networking on the guest.

**Note:** In the QEMU emulator, user mode networking is called *SLIRP*:

<http://bit.ly/lmvdVfT>

### Considerations

The main consideration of this scenario is that the virtual machine is not visible from the external network. For example, the virtual machine is able to access the Internet, but it will not be able to host an external accessible web server.

There are also two other considerations:

- ▶ There is significant overhead during the packet translation, which makes the performance very poor.
- ▶ Internet Control Message Protocol (ICMP) does not work with the user mode networking approach.

## 6.2.2 Network address translation networking

NAT networking uses a range of IPv4 private addresses in the guests and translates those addresses to a host address when routing to the external network. Guests that are using the same NAT network can communicate directly, as though they were on the same Ethernet broadcast domain.

## 6.2.3 Bridged networking

This is the approach that gives better performance than user mode, mainly because the guest network interface is exposed to the external network. Consequently, it can be accessed externally as with a non-virtualized system, as shown in Figure 6-1 on page 166.

A *network bridge* is an interface that connects other network segments that are described by IEEE 802.1D standard.

A bridge is capable of passing Layer 2 packets using the attached network interfaces. Because the packet forwarding works on Layer 2, any Layer 3 protocol works transparently.

The main limitation of bridged networking is that most wireless adapters do not accept it.

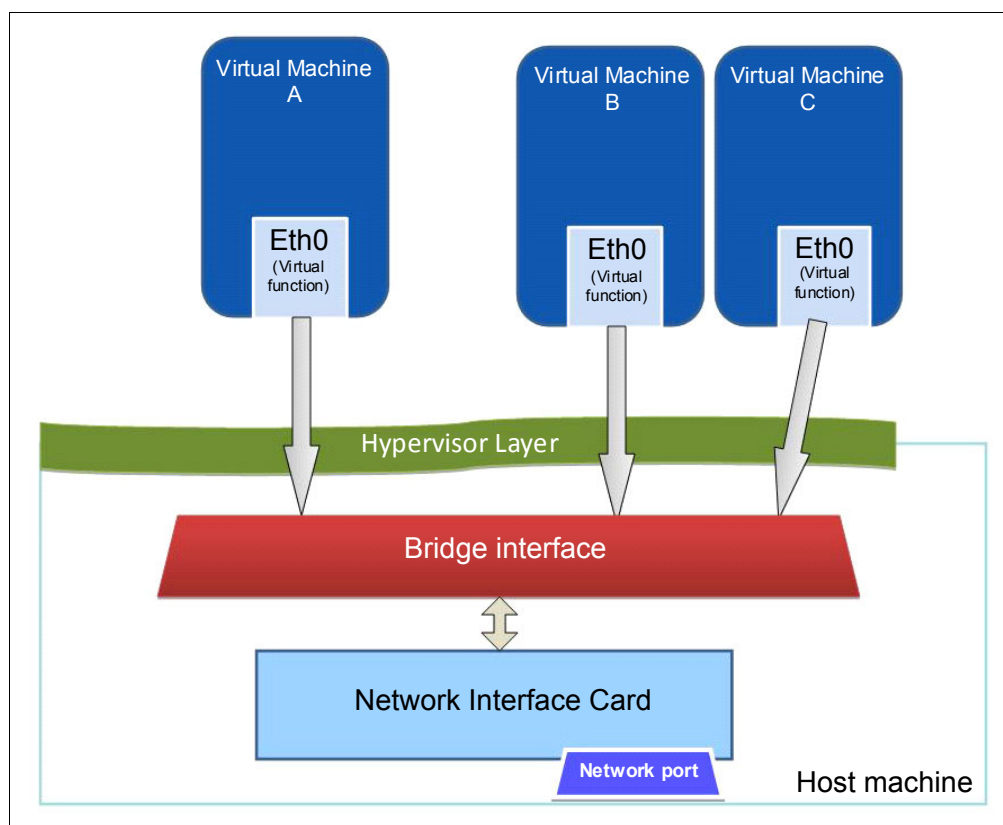


Figure 6-1 Bridge architecture

To manage the bridge architecture on the host, use the **brctl** command. For example, to create a bridge network interface, execute the following command:

```
# brctl addbr <example_bridge_device>
```

## 6.2.4 Open vSwitch

Open vSwitch is an open source, multilayer distributed virtual switch. It supports standard management protocols and interfaces, such as NetFlow, sFlow, SPAN, LACP, and 802.1ag. The key advantage of it is that Open vSwitch can be distributed across many physical machines. By providing APIs for automation, Open vSwitch becomes an almost perfect solution for OpenStack deployments.

If you already created a bridge with **linux-bridge-tools**, you can't reuse that bridge. The bridge must be re-created by using **ovs** tools.

It is required to have Open vSwitch service up and running before plugging virtual machines into the Open vSwitch network. See 4.3, "Manage guest networks" on page 114 for information about the initial configuration.

See the documentation on the Open vSwitch website for more information:

<http://openvswitch.org>

## 6.3 Storage pools

A *storage pool* is a pool of disks that is in the hypervisor and can be assigned to virtual machines. Different storage pools can be created and backed up by different technologies:

- ▶ File system directory
- ▶ Physical disk device
- ▶ Preformatted block device
- ▶ Internet Small Computer System Interface (iSCSI) target
- ▶ Logical Volume Manager (LVM) volume group
- ▶ Multipath devices
- ▶ Network exported directory
- ▶ Small Computer System Interface (SCSI) host adapter

### 6.3.1 Storage volume

A *storage volume* is a storage device that is usually assigned to a guest as a virtual disk. The virtual machine sees the device as a physical disk. The physical location of this block depends on the storage pool configuration.

Storage pools can be logically divided in two categories: Block device and file-backed pools.

### 6.3.2 Block device pools

Block device pools are pools such as the following kinds that reside on block devices:

- ▶ Logical volume pool

A virtual machine volume created on top of an LVM pool is an LVM logical volume.

- ▶ iSCSI volume pool

A volume in an iSCSI pool is a LUN presented by the iSCSI target, as shown in Figure 6-3 on page 169.

- ▶ SCSI volume pool

A volume in a Fibre Channel (FC) pool is a LUN that is mapped to a specific `scsi_host` adapter, as shown in Figure 6-5 on page 170.

- ▶ Disk volume pool

A volume in such a pool is a partition on the physical disk.

**Note:** You can't create new volumes on iSCSI and SCSI through a libvirt API. Volumes must be created manually on a target, instead.

### 6.3.3 File-backed pools

File-backed pools are file systems mounted to a PowerKVM host. There are two types: File system pools and network file system pools.

#### File system pool

A volume in a file system storage pool is an image file that is stored in a specified format, as shown in Figure 6-2.

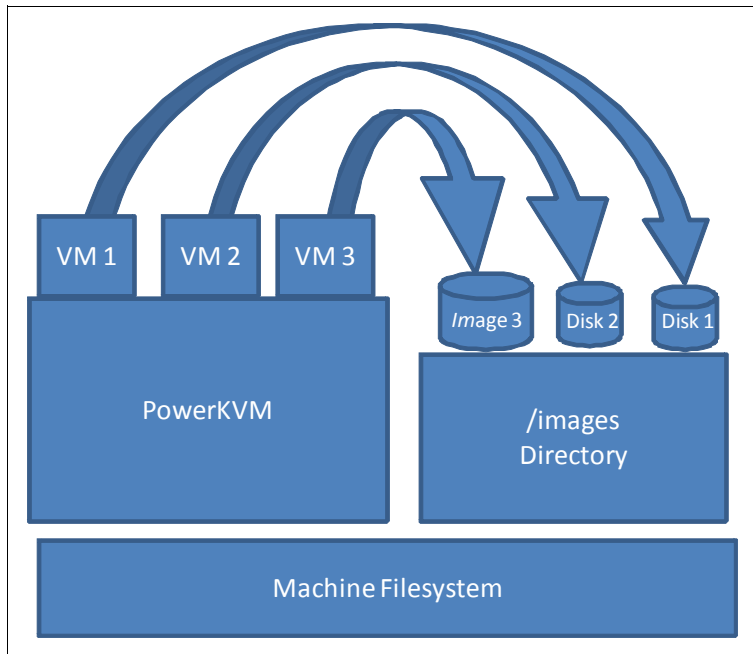


Figure 6-2 File-backed pool

The file system can be backed by iSCSI, as shown in Figure 6-3 on page 169.



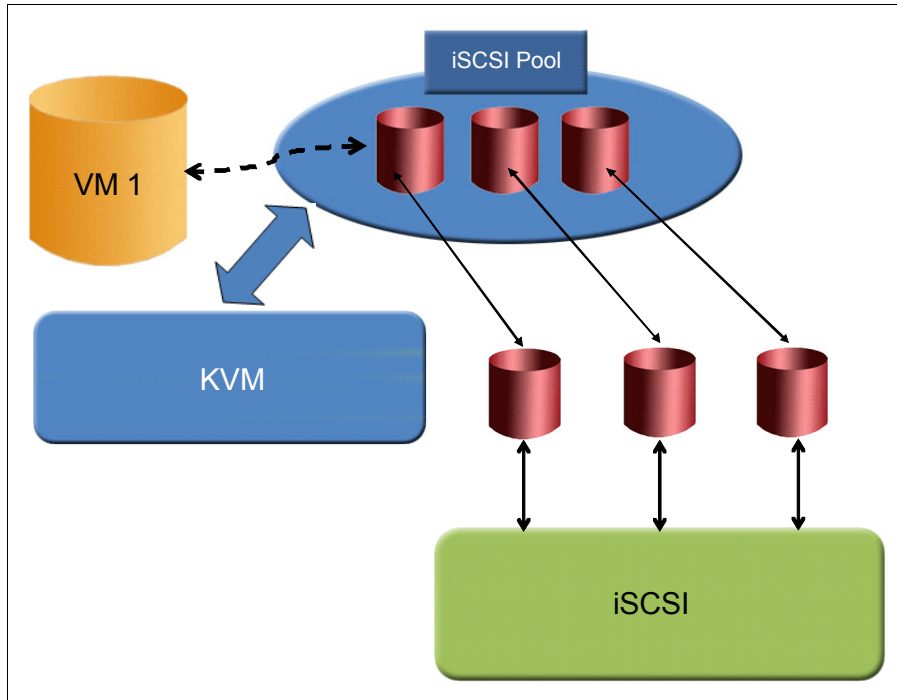


Figure 6-3 iSCSI-backed storage pool

### Network file system pool

This is nearly the same as a file system pool, but it is hosted on a remote file system (NFS), as shown in Figure 6-4.

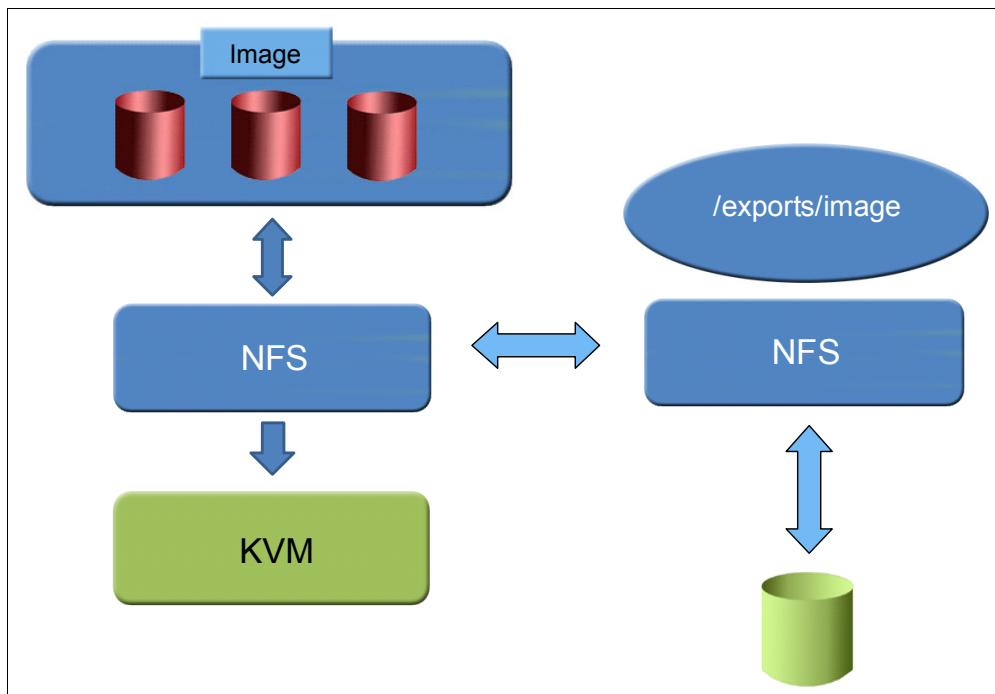


Figure 6-4 NFS-backed storage device

Either file system type can be on iSCS or Fibre Channel, as shown in Figure 6-5.

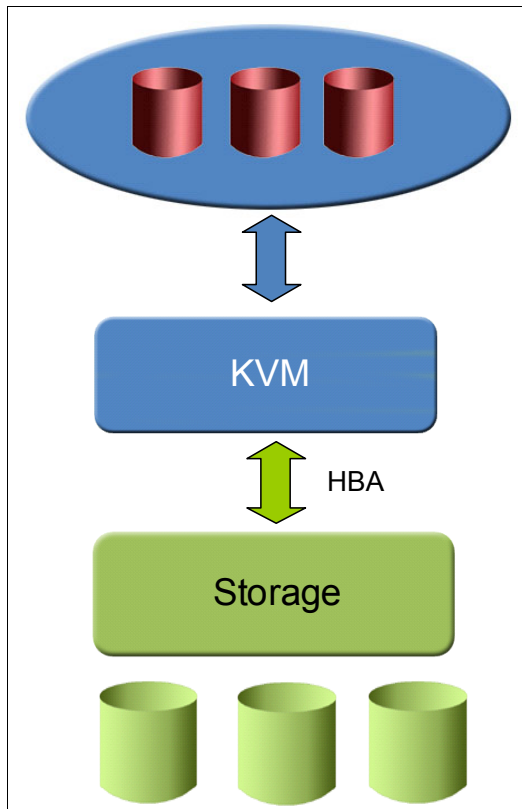


Figure 6-5 Fibre Channel-backed storage device

## 6.4 I/O pass-through

I/O pass-through is a method to give to the virtual machines the I/O adapters that are plugged into the host machine. There are three types:

- ▶ SCSI pass-through
- ▶ USB pass-through
- ▶ PCI pass-through

### 6.4.1 SCSI pass-through

By default, QEMU is between the virtual machine and the physical block device. It provides a general SCSI device to a virtual machine. If you want to connect a pass-through SCSI device directly to a VM, set `device='lun'` as Example 6-1 shows.

*Example 6-1 LUN pass-through*

---

```
<disk type='block' device='lun'>
  <driver name='qemu' type='raw'/>
  <source dev='/dev/sdn'/>
  <target dev='sda' bus='scsi'/>
  <address type='drive' controller='0' bus='0' target='3' unit='0'/>
</disk>
```

---

## 6.4.2 USB pass-through

It is also possible to pass through a USB host device to a virtual machine that is running under PowerKVM. First, determine the vendor and ID values of the device by using the `lsusb` command. The output is shown in Example 6-2.

*Example 6-2 lsusb command output*

---

```
[root@powerkvm ~]# lsusb
Bus 001 Device 003: ID 058f:6387 Alcor Micro Corp. Flash Drive
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 003 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
```

---

**Note:** There are limitations, though:

- ▶ Only USB 2.0 and USB 3.0 devices work for USB pass-through.
- ▶ The device must not be used by the host system at the same time that the virtual machine is using it. To prevent that, kernel modules must be unloaded with the `rmmod` command.
- ▶ If the device is a flash drive, the only requirement is to not have it mounted by host.

You must prepare an XML file with a device description. There are two options available:

- ▶ Specify the device by *vendor, product* pair, for example 058f:6387 (see Example 6-3).
- ▶ Specify the device by *bus, device* pair, for example 001:003 (see Example 6-4).

**Tips:** We discovered during testing that vendor, product definition works only with a cold plug (the virtual machine is in shut-off state). The bus, device combination works quite well for a hot plug.

*Example 6-3 USB XML description example based on vendor, product (IDs) pair*

---

```
<hostdev type='usb'>
  <source>
    <vendor id='0x058f' />
    <product id='0x6387' />
  </source>
</hostdev>
```

---

*Example 6-4 USB XML description example based on bus, device pair*

---

```
<hostdev type='usb'>
  <source>
    <address bus='1' device='3' />
  </source>
</hostdev>
```

---

Example 6-5 shows how to check what is connected to the virtual machine.

*Example 6-5 lsusb command output from a virtual machine*

---

```
sles11vm01:~ # lsusb
Bus 001 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
```

---

When you are ready to add the device, you can plug it in by using the **virsh attach-device** command, as shown in Example 6-6.

*Example 6-6 Attach the device to a VM*

---

```
# virsh attach-device sles11vm01 usb-bus.xml
Device attached successfully

# virsh console sles11vm01
Connected to domain sles11vm01
Escape character is ^]

# lsusb
Bus 001 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 001 Device 003: ID 058f:6387 Alcor Micro Corp. Transcend JetFlash Flash Drive
```

---

**Note:** To make a live change persistent, use the **--persistent** or **--config** option.

To detach a USB device, use the **virsh detach-device** command, as shown in Example 6-7.

*Example 6-7 Detaching a USB device from a guest*

---

```
# virsh detach-device sles11vm01 usb-bus.xml
Device detached successfully

# virsh console sles11vm01
Connected to domain sles11vm01
Escape character is ^]

# lsusb
Bus 001 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
sles11vm01:~ #
```

---

### 6.4.3 PCI pass-through to a virtual machine

To pass through a specific adapter to a virtual machine, the adapter first needs to be described in an XML-formatted file. You can use either **lspci** (see Example 6-8) or **virsh nodedev-list | grep pci** (see Example 6-9 on page 173) to find the device address of a card that you want to pass through.

*Example 6-8 lspci command output*

---

```
# lspci
0000:00:00.0 PCI bridge: IBM Device 03b9 (rev 10)
0000:01:00.0 PCI bridge: PLX Technology, Inc. PEX 8624 24-lane, 6-Port PCI Express
Gen 2 (5.0 GT/s) Switch [ExpressLane] (rev bb)
0000:02:04.0 PCI bridge: PLX Technology, Inc. PEX 8624 24-lane, 6-Port PCI Express
Gen 2 (5.0 GT/s) Switch [ExpressLane] (rev bb)
0000:02:05.0 PCI bridge: PLX Technology, Inc. PEX 8624 24-lane, 6-Port PCI Express
Gen 2 (5.0 GT/s) Switch [ExpressLane] (rev bb)
0000:02:06.0 PCI bridge: PLX Technology, Inc. PEX 8624 24-lane, 6-Port PCI Express
Gen 2 (5.0 GT/s) Switch [ExpressLane] (rev bb)
0000:02:08.0 PCI bridge: PLX Technology, Inc. PEX 8624 24-lane, 6-Port PCI Express
Gen 2 (5.0 GT/s) Switch [ExpressLane] (rev bb)
```

---

```

0000:02:09.0 PCI bridge: PLX Technology, Inc. PEX 8624 24-lane, 6-Port PCI Express
Gen 2 (5.0 GT/s) Switch [ExpressLane] (rev bb)
0000:40:00.0 RAID bus controller: IBM Obsidian-E PCI-E SCSI controller (rev 01)
0000:60:00.0 RAID bus controller: IBM Obsidian-E PCI-E SCSI controller (rev 01)
0000:80:00.0 PCI bridge: PLX Technology, Inc. PEX8112 x1 Lane PCI Express-to-PCI
Bridge (rev aa)
0000:90:01.0 USB controller: NEC Corporation OHCI USB Controller (rev 43)
0000:90:01.1 USB controller: NEC Corporation OHCI USB Controller (rev 43)
0000:90:01.2 USB controller: NEC Corporation uPD72010x USB 2.0 Controller (rev 04)
0000:a0:00.0 Ethernet controller: Broadcom Corporation NetXtreme BCM5719 Gigabit
Ethernet PCIe (rev 01)
0000:a0:00.1 Ethernet controller: Broadcom Corporation NetXtreme BCM5719 Gigabit
Ethernet PCIe (rev 01)
0000:a0:00.2 Ethernet controller: Broadcom Corporation NetXtreme BCM5719 Gigabit
Ethernet PCIe (rev 01)
0000:a0:00.3 Ethernet controller: Broadcom Corporation NetXtreme BCM5719 Gigabit
Ethernet PCIe (rev 01)
0001:00:00.0 PCI bridge: IBM Device 03b9 (rev 10)
0001:01:00.0 Ethernet controller: Mellanox Technologies MT26448 [ConnectX EN
10GigE, PCIe 2.0 5GT/s] (rev b0)
0002:00:00.0 PCI bridge: IBM Device 03b9 (rev 10)
0002:01:00.0 RAID bus controller: IBM Obsidian-E PCI-E SCSI controller (rev 01)
0003:00:00.0 PCI bridge: IBM Device 03b9 (rev 10)
0003:01:00.0 RAID bus controller: IBM Obsidian-E PCI-E SCSI controller (rev 01)
0004:00:00.0 PCI bridge: IBM Device 03b9 (rev 10)
0004:01:00.0 RAID bus controller: IBM Obsidian-E PCI-E SCSI controller (rev 01)
0005:00:00.0 PCI bridge: IBM Device 03b9 (rev 10)
0005:01:00.0 RAID bus controller: IBM Obsidian-E PCI-E SCSI controller (rev 01)

```

---

*Example 6-9 virsh nodedev-list command output*

```

# virsh nodedev-list | grep pci
pci_0000_00_00_0
pci_0000_01_00_0
pci_0000_02_04_0
pci_0000_02_05_0
pci_0000_02_06_0

```

---

The next example shows passing through a Mellanox adapter. As you can see in Example 6-10, the device has a 0001:01:00.0 PCI address, and **virsh** represents it as pci\_0001\_01\_00\_0. To make sure that you have found the correct match, using **virsh nodedev-dumpxml pci\_0001\_01\_00\_0** provides more information about the device, as shown in Example 6-10.

*Example 6-10 virsh nodedev-dumpxml command output*

```

# virsh nodedev-dumpxml pci_0001_01_00_0
<device>
  <name>pci_0001_01_00_0</name>
  <path>/sys/devices/pci0001:00/0001:00:00.0/0001:01:00.0</path>
  <parent>pci_0001_00_00_0</parent>
  <driver>
    <name>m1x4_core</name>
  </driver>
  <capability type='pci'>
    <domain>1</domain>
  </capability>
</device>

```

```

<bus>1</bus>
<slot>0</slot>
<function>0</function>
<product id='0x6750'>MT26448 [ConnectX EN 10GigE, PCIe 2.0 5GT/s]</product>
<vendor id='0x15b3'>Mellanox Technologies</vendor>
<iommuGroup number='4'>
  <address domain='0x0001' bus='0x01' slot='0x00' function='0x0' />
</iommuGroup>
</capability>
</device>

```

---

Next, the PCI adapter needs to be detached from a host system. You can use the **virsh nodedev-detach** command to do that (see Example 6-11).

*Example 6-11 virsh nodedev-detach command*

```

# virsh nodedev-detach pci_0001_01_00_0
Device pci_0001_01_00_0 detached

```

---

After the PCI adapter is detached from the system, the adapter needs to be described in a virtual machine configuration. To put the adapter description in the *<devices>* section, edit the virtual machine configuration by using the **virsh edit** command (see Example 6-12). Then, save the file, and the machine is ready to be started.

*Example 6-12 PCI adapter description*

```

# virsh edit sles11vm01

<hostdev mode='subsystem' type='pci' managed="yes">
<source>
<address domain='0x0001' bus='0x01' slot='0x00' function='0x00' />
</source>
<driver name='vfio' />
</hostdev>

```

---

**Note:** The status of the managed mode can be either “yes” or “no”.

- ▶ yes: Libvirt will unbind the device from the existing driver, reset the device, and bind it to pci-stub
- ▶ no: You must take care about those aspects manually

Now, you are ready to start the guest with the assigned adapter. In Example 6-13, you can see that the virtual machine detects the card correctly. Example 6-14 on page 175 shows that the additional Ethernet interfaces are available.

*Example 6-13 lspci command output within the VM*

```

sles11vm01:~ # lspci
0000:00:01.0 Ethernet controller: Red Hat, Inc Virtio network device
0000:00:02.0 USB controller: Apple Inc. KeyLargo/Intrepid USB
0001:00:00.0 Ethernet controller: Mellanox Technologies MT26448 [ConnectX EN
10GigE, PCIe 2.0 5GT/s] (rev b0)

```

---

Example 6-14 List of available interfaces

---

```
# ip a s
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.255.255.255 scope host lo
    inet 127.0.0.2/8 brd 127.255.255.255 scope host secondary lo
    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 qlen 1000
    link/ether 52:54:00:de:18:a2 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.213/24 brd 192.168.122.255 scope global eth0
    inet6 fe80::5054:ff:fede:18a2/64 scope link
        valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN qlen 1000
    link/ether 00:02:c9:2b:98:00 brd ff:ff:ff:ff:ff:ff
4: eth2: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN qlen 1000
    link/ether 00:02:c9:2b:98:01 brd ff:ff:ff:ff:ff:ff
```

---

## 6.4.4 I/O limits

It is possible to override certain properties of a block device, such as disk geometry or logical and physical block sizes. Those tunables might be especially useful during development and testing of software on a PowerKVM virtual machine. Example 6-15 shows an example of how to use `<blockio>`, `<geometry>`, and `<target>` within the `<disk>` element.

Example 6-15 Change disk properties

---

```
<disk type='block' device='disk'>
  <driver name='qemu' type='raw'/>
  <source dev='/dev/sdn'/>
  <target dev='sda' bus='scsi'/>
  <geometry culs='16383' heads='16' secs='63' trans='1ba'/>
  <blockio logical_block_size='512' physical_block_size='4096'>
</disk>
```

---

It is also possible to limit the I/O throughput of a device with the `<iotune>` subelement of a `<disk>` element. These are the options:

**total\_bytes\_sec**      Total throughput limit in bytes per second

**Note:** `total_bytes_sec` conflicts with `read_bytes_sec` and `write_bytes_sec`.

**read\_bytes\_sec**      Read throughput in bytes per second

**write\_bytes\_sec**      Write throughput in bytes per second

**total\_iops\_sec**      Total I/O operations per second

**Note:** `total_iops_sec` conflicts with `read_iops_sec` and `write_iops_sec`.

**read\_iops\_sec**      Read I/O operations per second

**write\_iops\_sec**      Write I/O operations per second

Example 6-16 shows the complete <disk> element of a virtual machine.

*Example 6-16 iotune example of a disk element*

---

```
<disk type='file' device='disk'>
  <driver name='qemu' type='qcow2' cache='none' />
  <source file='/var/lib/libvirt/images/sles11vm04.qcow2' />
  <target dev='sda' bus='scsi' />
  <address type='drive' controller='0' bus='0' target='0' unit='0' />
  <iotune>
    <total_bytes_sec>10000000</total_bytes_sec>
    <read_iops_sec>400000</read_iops_sec>
    <write_iops_sec>100000</write_iops_sec>
  </iotune>
</disk>
```

---

## 6.5 N\_Port ID Virtualization (NPIV)

N\_Port ID Virtualization (NPIV) is a technique to virtualize Fibre Channel adapters. It allows creating a virtual adapter that is connected through the physical (parent) adapter to the SAN fabric. In this section, we show how to create a virtual Fibre Channel adapter and how to use it.

**Important:** NPIV is available in PowerKVM V3.1 as a technology preview only. Further development is pending.

Before creating a virtual Fibre Channel adapter, first discover the adapters in your system, as described in Example 6-17.

*Example 6-17 Discovery of Fibre Channel adapters*

---

```
# virsh nodedev-list --cap vports
scsi_host1
scsi_host2

# virsh nodedev-dumpxml scsi_host1
<device>
  <name>scsi_host1</name>

  <path>/sys/devices/pci0001:00/0001:00:00.0/0001:01:00.0/0001:02:09.0/0001:09:00.0/
host1</path>
  <parent>pci_0001_09_00_0</parent>
  <capability type='scsi_host'>
    <host>1</host>
    <unique_id>0</unique_id>
    <capability type='fc_host'>
      <wwnn>20000120fa89ca40</wwnn>
      <wwpn>10000090fa89ca40</wwpn>
      <fabric_wwn>100000053345e69e</fabric_wwn>
    </capability>
    <capability type='vport_ops'>
      <max_vports>255</max_vports>
      <vports>0</vports>
    </capability>
```



```

    </capability>
</device>

```

```

# virsh nodedev-dumpxml scsi_host2

```

```

<device>
  <name>scsi_host2</name>

  <path>/sys/devices/pci0001:00/0001:00:00.0/0001:01:00.0/0001:02:09.0/0001:09:00.1/
host2</path>
  <parent>pci_0001_09_00_1</parent>
  <capability type='scsi_host'>
    <host>2</host>
    <unique_id>1</unique_id>
    <capability type='fc_host'>
      <wwnn>20000120fa89ca41</wwnn>
      <wwpn>10000090fa89ca41</wwpn>
      <fabric_wwn>0</fabric_wwn>
    </capability>
    <capability type='vport_ops'>
      <max_vports>255</max_vports>
      <vports>0</vports>
    </capability>
  </capability>
</device>

```

---

In Example 6-17 on page 176, you can see that this system has two Fibre Channel ports (on one 2-port adapter). Both ports are able to support up to 255 virtual ports or NPIV ports. In the example, you also can see the worldwide port names (WWPNs) of the adapters and the fabric worldwide names (WWNs). In this example, WWN 0 shows that the second port (scsi\_host2) has no connection to a fabric.

In the example above, we use the first port (*scsi\_host1*) to create a virtual Fibre Channel port. To create a virtual Fibre Channel adapter, you need an XML snippet that refers to the parent adapter as shown in Example 6-18. The example also shows how to create the adapter using *virsh nodedev-create* and what the attributes of the new adapter look like.

---

*Example 6-18 Creating a virtual Fibre Channel adapter*

---

```

# cat vfc.xml

```

```

<device>
  <parent>scsi_host1</parent>
  <capability type='scsi_host'>
    <capability type='fc_host'>
      </capability>
    </capability>
  </device>

```

```

# virsh nodedev-create vfc.xml

```

```

Node device scsi_host3 created from vfc.xml

```

```

# virsh nodedev-dumpxml scsi_host3

```

```

<device>
  <name>scsi_host3</name>

```

```
<path>/sys/devices/pci0001:00/0001:00:00.0/0001:01:00.0/0001:02:09.0/0001:09:00.0/
host1/vport-1:0-0/host3</path>
  <parent>scsi_host1</parent>
  <capability type='scsi_host'>
    <host>3</host>
    <unique_id>2</unique_id>
    <capability type='fc_host'>
      <wwnn>5001a4a3b41775fd</wwnn>
      <wwpn>5001a4a4b5723b96</wwpn>
      <fabric_wwn>100000053345e69e</fabric_wwn>
    </capability>
  </capability>
</device>
```

In Example 6-18 on page 177, you can also see the assigned WWPN for the virtual Fibre Channel adapter and that it is connected to the same fabric as the parent adapter.

NPIV is a technology preview and not supported by Kimchi yet, but Kimchi also shows the virtual Fibre Channel adapters as shown in Figure 6-6.

Host	Guests	Templates	Storage	Network	Administration																								
<div> <div>Firmware Update</div> <div>Configuration Backup</div> <div>Network Configuration</div> <div>Power Options</div> <div> <div>SAN Adapters</div> <table> <tr> <th>Name</th><th>WWPN</th><th>WWNN</th><th>State</th><th>In-Use/Max Ports</th><th>Speed</th></tr> <tr> <td>host1</td><td>0x10000090fa89ca40</td><td>0x20000120fa89ca40</td><td>Online</td><td>1/255</td><td>4 Gbit</td></tr> <tr> <td>host2</td><td>0x10000090fa89ca41</td><td>0x20000120fa89ca41</td><td>Linkdown</td><td>0/255</td><td>unknown</td></tr> <tr> <td>host3</td><td>0x5001a4a4b5723b96</td><td>0x5001a4a3b41775fd</td><td>Online</td><td>Unknown/Unknown</td><td>4 Gbit</td></tr> </table> <div> <div>Symbolic Name</div> <div>Emulex 00E0806 FV2.02X7 DV10.4.8000.0. HN:localhost OS:Linux</div> <div>Emulex 00E0806 FV2.02X7 DV10.4.8000.0. HN:localhost OS:Linux</div> <div>Emulex 00E0806 FV2.02X7 DV10.4.8000.0. HN:s812Hvmm-test OS:Linux</div> </div> </div> </div> <div>Sensor Monitor</div> <div>SEP Configuration</div> <div>User Management</div>						Name	WWPN	WWNN	State	In-Use/Max Ports	Speed	host1	0x10000090fa89ca40	0x20000120fa89ca40	Online	1/255	4 Gbit	host2	0x10000090fa89ca41	0x20000120fa89ca41	Linkdown	0/255	unknown	host3	0x5001a4a4b5723b96	0x5001a4a3b41775fd	Online	Unknown/Unknown	4 Gbit
Name	WWPN	WWNN	State	In-Use/Max Ports	Speed																								
host1	0x10000090fa89ca40	0x20000120fa89ca40	Online	1/255	4 Gbit																								
host2	0x10000090fa89ca41	0x20000120fa89ca41	Linkdown	0/255	unknown																								
host3	0x5001a4a4b5723b96	0x5001a4a3b41775fd	Online	Unknown/Unknown	4 Gbit																								

Figure 6-6 Virtual Fibre Channel adapters in Kimchi

After zoning the new WWPN to your storage and assigning a disk to it, you can scan for new disks by using the following command:

```
# rescan-scsi-bus.sh -a
```

**Important:** Use the flag -a, otherwise the new virtual Fibre Channel adapter will not be scanned and the disk/LUN will not be added.

Example 6-19 on page 179 shows the new discovered disk. There are many ways to show the new device, in this case, by using the `virsh nodedev-list` command. The new SCSI device appears under `scsi_host3`, which is our just created virtual Fibre Channel adapter. The new device appears twice because there are two paths to that device. For more information about multipathing, see 6.6, “Using multipath disks” on page 182.

```
+-- net_lo_00_00_00_00_00_00
+- net_virbr0_nic_52_54_00_7e_19_0b
+- net_virbr1_nic_52_54_00_49_7f_ca
+- net_vnet0_fe_54_00_6d_f9_bf
+- pci_0000_00_00_0
+- pci_0001_00_00_0
|
|-- pci_0001_01_00_0
|   |
|   |-- pci_0001_02_01_0
|   |-- pci_0001_02_08_0
|       |
|       |-- pci_0001_08_00_0
|           |
|           |-- scsi_host0
|               |
|               +- scsi_target0_0_0
|
+-- pci_0001_02_09_0
    |
    |-- pci_0001_09_00_0
        |
        |-- scsi_host1
            |
            +- scsi_host3
                |
                |-- scsi_target3_0_0
                    |
                    +- scsi_3_0_0_0
                        |
                        +- block_sdk_3600507680281038b08000000000000a8
                        +- scsi_generic_sg22
                |
                |-- scsi_target3_0_1
                    |
                    +- scsi_3_0_1_0
                        |
                        +- block_sd1_3600507680281038b08000000000000a8
                        +- scsi_generic_sg23
            |
            +- scsi_target1_0_0
                |
                +- scsi_1_0_0_0
                    |
                    +- block_sde_3600507680281038b080000000000008b
                    +- scsi generic sg16
```

Now that the disk is available, you can assign it to a guest as a multipath device, which is explained in section 6.6.2, “Direct mapped multipath disks” on page 185. But the virtual Fibre Channel adapter that we created as explained in this section is not persistent yet. If you reboot the host then, the definition of the virtual Fibre Channel adapter would be lost.

To make the adapter persistent, the preferred practice is to create a storage pool using this new NPIV adapter. To create a storage pool, an XML snippet with the WWNN and WWPN is needed. To get the two numbers, use the **virsh nodedev-dumpxml** command as explained in Example 6-18 on page 177.

Example 6-20 shows the steps to be done in order to create the storage pool and make the adapter persistent.

*Example 6-20 Creating a storage pool for a virtual Fibre Channel adapter*

---

```
# cat vfcpool.xml
<pool type='scsi'>
  <name>vfcpool</name>
  <source>
    <adapter type='fc_host' wwnn='5001a4a3b41775fd' wwpn='5001a4a4b5723b96' />
  </source>
  <target>
    <path>/dev/disk/by-path</path>
    <permissions>
      <mode>0700</mode>
      <owner>0</owner>
      <group>0</group>
    </permissions>
  </target>
</pool>

# virsh pool-define vfcpool.xml
Pool vfcpool defined from vfcpool.xml

# virsh pool-start vfcpool
Pool vfcpool started

# virsh vol-list vfcpool
Name                                     Path
-----
unit:0:0:0
/dev/disk/by-path/pci-0001:09:00.0-fc-0x5005076802109670-lun-0
unit:0:1:0
/dev/disk/by-path/pci-0001:09:00.0-fc-0x5005076802109671-lun-0
```

---

After you created the storage pool, the volumes of the pool can be assigned to a guest by using the XML snippet as shown in Example 6-21 or by using Kimchi.

*Example 6-21 Attachment of a disk from a storage pool on a virtual Fibre Channel adapter*

---

```
[powerkvm-host]# cat add_pool_lun.xml
<disk type='volume' device='disk'>
  <driver name='qemu' type='raw' />
  <source pool='vfcpool' volume='unit:0:0:0' />
  <target dev='sda' bus='scsi' />
</disk>
```

```
[linux-guest]# fdisk -l
Disk /dev/vda: 10 GiB, 10737418240 bytes, 20971520 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x00032025
```

Device	Boot	Start	End	Sectors	Size	Id	Type
/dev/vda1	*	2048	4095	2048	1M	41	PPC PReP Boot
/dev/vda2		4096	4225023	4220928	2G	82	Linux swap / Solaris
/dev/vda3		4225024	20971519	16746496	8G	83	Linux

```
Disk /dev/sda: 40 GiB, 42949672960 bytes, 83886080 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

---

If the two volumes, as shown in Example 6-20 on page 180, are two paths of the same volume, the preferred practice is to assign a multipath device (mpath) to the guest and not just one or both units of the storage pool. If both units are assigned to the guest, the operating system in the virtual machine is not aware that the two disks are just one disk with two paths, as the disks appear as virtualized QEMU disks. To attach a multipath device, follow the steps in section 6.6.2, “Direct mapped multipath disks” on page 185.

## 6.6 Using multipath disks

Multipath disks are disks or volumes on a SAN storage that can be addressed over different paths, for example in different Fibre Channel switch fabrics. Figure 6-7 shows a typical environment with two paths to the storage, using two host bus adapters (HBAs) connected to two SAN fabrics for redundancy and performance reasons.

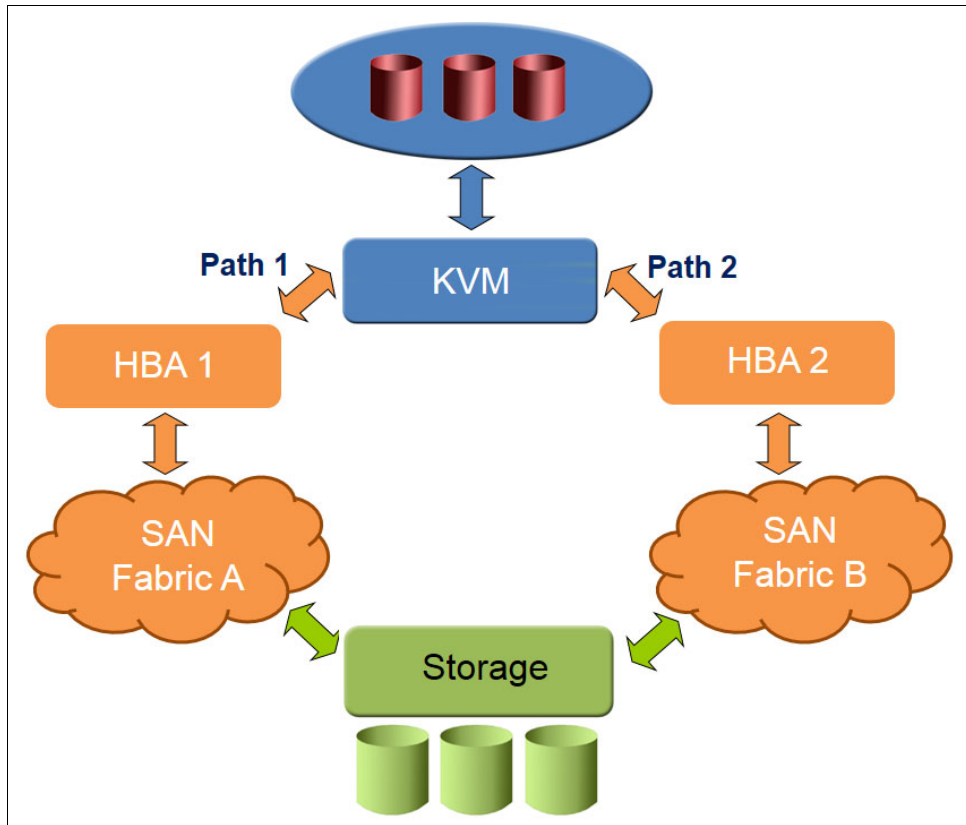


Figure 6-7 Typical multipath environment

The following sections describe how to handle multipath disks and how these can be used with PowerKVM.

### 6.6.1 Multipath disk handling

There are several ways to work with multipath disks. The most common command for handling multipath disks is *multipath*. Example 6-22 shows three disks that have two paths each. These disks are named mpatha to mpathc. The disks are 10 G, 25 G, and 50 G in size and all paths are in active state.

Example 6-22 Example for multipath disk

```
# multipath -ll
mpathc (3600507680281038b0800000000000a7) dm-6 IBM      ,2145
size=50G features='1 queue_if_no_path' hwhandler='0' wp=rw
|-+- policy='service-time 0' prio=50 status=active
|  ~ 1:0:0:2 sdg 8:96  active ready running
`-+- policy='service-time 0' prio=10 status=enabled
    ~ 1:0:1:2 sdj 8:144 active ready running
```

```

mpathb (3600507680281038b0800000000000a5) dm-5 IBM      ,2145
size=25G features='1 queue_if_no_path' hwhandler='0' wp=rw
|+- policy='service-time 0' prio=50 status=active
|  ~- 1:0:1:1 sdi 8:128 active ready running
|+- policy='service-time 0' prio=10 status=enabled
|  ~- 1:0:0:1 sdf 8:80  active ready running
mpatha (3600507680281038b08000000000008b) dm-4 IBM      ,2145
size=10G features='1 queue_if_no_path' hwhandler='0' wp=rw
|+- policy='service-time 0' prio=50 status=active
|  ~- 1:0:0:0 sde 8:64  active ready running
|+- policy='service-time 0' prio=10 status=enabled
|  ~- 1:0:1:0 sdh 8:112 active ready running

```

In Example 6-22 on page 182, you can also see the UUIDs of the LUNs in the storage. This example is taken from an IBM Storwize® V7000. Figure 6-8 shows the disks (LUNs) on the storage with the corresponding UUIDs.

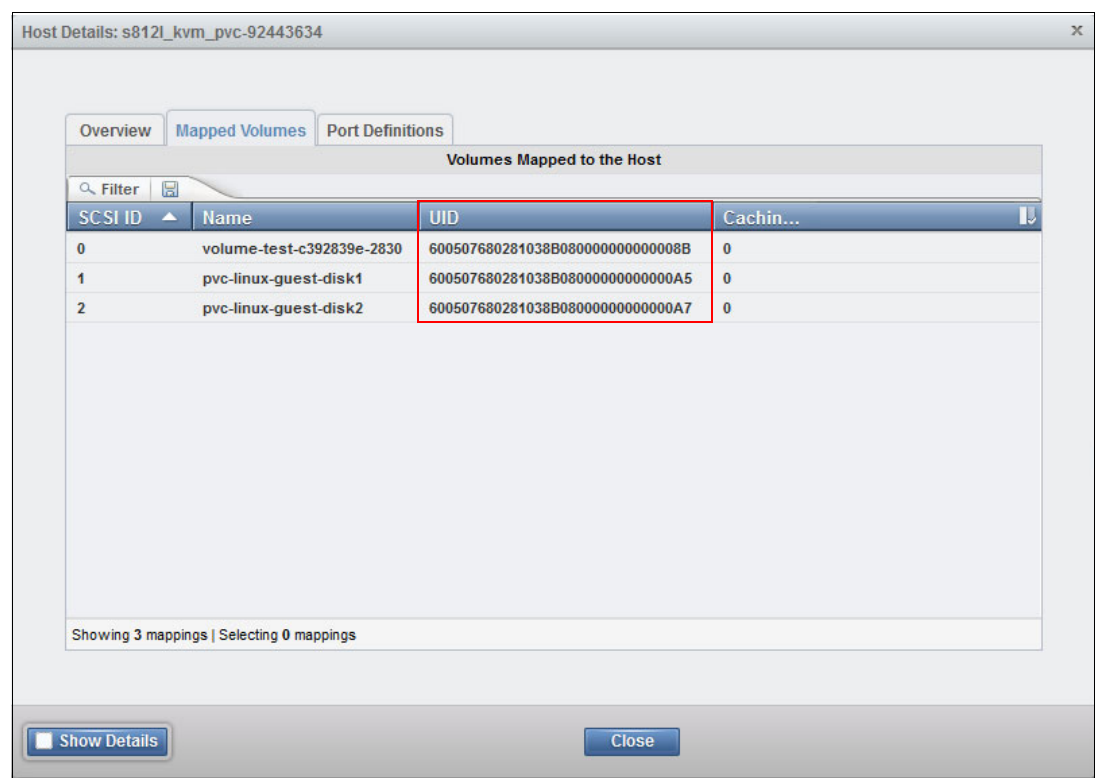


Figure 6-8 IBM Storwize V7000 storage view of LUNs attached to PowerKVM

There are many attributes of multipath disks that can be changed in the `/etc/multipath.conf` file. Example 6-23 shows some changes that can be made. The defaults for the first LUN ending on “8b” are changed to a Round Robin path algorithm with all paths equal in one priority group and an alias name *pvc-disk*. The second LUN ending on “a7” should only get an alias name *pkvm-pool1*.

*Example 6-23 Changes in `/etc/multipath.conf`*

---

```
defaults {
    find_multipaths yes
    user_friendly_names yes
}

blacklist {
}

multipaths {
    multipath {
        wwid "3600507680281038b080000000000008b"
        path_grouping_policy multibus
        path_selector          "round-robin 0"
        alias pvc-disk
    }

    multipath {
        wwid "3600507680281038b08000000000000a7"
        alias pkvm-pool1
    }
}
```

---

After changing the attributes, the multipath service needs to be restarted with:

```
# service multipathd stop
# service multipathd start
```

**Note:** There are more attributes that can be changed. The changed values above are just an example to illustrate how this can be achieved.

After these changes, the output of `multipath -ll` looks as shown in Example 6-24.

*Example 6-24 Output of `multipath -ll` after some changes*

---

```
# multipath -ll
pkvm-pool1 (3600507680281038b08000000000000a7) dm-5 IBM      ,2145
size=50G features='0' hwhandler='0' wp=rw
|+- policy='service-time 0' prio=50 status=active
|  ~ 1:0:1:2 sdj 8:144 active ready running
|+- policy='service-time 0' prio=10 status=enabled
|  ~ 1:0:0:2 sdg 8:96  active ready running
mpathb (3600507680281038b08000000000000a5) dm-6 IBM      ,2145
size=25G features='0' hwhandler='0' wp=rw
|+- policy='service-time 0' prio=50 status=active
|  ~ 1:0:0:1 sdf 8:80  active ready running
|+- policy='service-time 0' prio=10 status=enabled
|  ~ 1:0:1:1 sdi 8:128 active ready running
pvc-disk (3600507680281038b080000000000008b) dm-4 IBM      ,2145
size=10G features='0' hwhandler='0' wp=rw
```



```

~+- policy='round-robin 0' prio=30 status=active
| 1:0:0:0 sde 8:64 active ready running
~- 1:0:1:0 sdh 8:112 active ready running

```

---

If a path fails, for example as shown in Example 6-25, depending on the multipath configuration there might be some recovery steps after bringing the path back online again. In the default configuration, the path shows up active automatically after the connection is restored.

*Example 6-25 Example of a path failure*

```

# multipath -ll
pkvm-pool1 (3600507680281038b08000000000000a7) dm-5 IBM      ,2145
size=50G features='0' hwhandler='0' wp=rw
|+- policy='service-time 0' prio=50 status=active
| ~- 1:0:1:2 sdj 8:144 active ready running
~+- policy='service-time 0' prio=10 status=enabled
  ~- 1:0:0:2 sdg 8:96 active ready running
mpathb (3600507680281038b08000000000000a5) dm-6 IBM      ,2145
size=25G features='0' hwhandler='0' wp=rw
|+- policy='service-time 0' prio=50 status=active
| ~- 1:0:0:1 sdf 8:80  failed faulty offline
~+- policy='service-time 0' prio=10 status=enabled
  ~- 1:0:1:1 sdi 8:128 active ready running
pvc-disk (3600507680281038b080000000000008b) dm-4 IBM      ,2145
size=10G features='0' hwhandler='0' wp=rw
~+- policy='round-robin 0' prio=30 status=active
| 1:0:0:0 sde 8:64 active ready running
~- 1:0:1:0 sdh 8:112 active ready running

```

---

**Note:** If a path is not used, it might still be shown as active although the path is failed. After the first I/O to a failed path, it shows up as failed.

If new disks were added to the PowerKVM host, these do not show up automatically. To scan for new devices, use the following command:

```
# rescan-scsi-bus.sh
```

## 6.6.2 Direct mapped multipath disks

A multipath disk can be directly mapped into a guest by using an XML snippet, as shown in Example 6-26.

*Example 6-26 XML snippet for attachment of a multipath disk into a guest*

```

# cat mpio_disk.xml
<disk type='block' device='disk'>
  <driver name='qemu' type='raw' />
  <source dev='/dev/disk/by-id/dm-name-mpathb' />
  <target dev='sda' bus='scsi' />
</disk>

# virsh attach-device linux-guest mpio_disk.xml --live --config
Device attached successfully

```

---

After running the **rescan-scsi-bus.sh** command on the guest, the disk shows up as shown in Example 6-27.

*Example 6-27 Mapped multipath disk inside the guest*

---

```
# ls -al /dev/disk/by-id
total 0
drwxr-xr-x 2 root root 60 Nov 12 13:44 .
drwxr-xr-x 6 root root 120 Nov 12 13:44 ..
lrwxrwxrwx 1 root root 9 Nov 12 13:44 scsi-0QEMU_QEMU_CD-ROM_drive-scsi0-0-0-2
-> ../../sr0

# rescan-scsi-bus.sh
Scanning SCSI subsystem for new devices
Scanning host 0 for SCSI target IDs 0 1 2 3 4 5 6 7, all LUNs
...

# lsscsi
[0:0:0:0] disk QEMU QEMU HARDDISK 2.3. /dev/sda
[0:0:0:2] cd/dvd QEMU QEMU CD-ROM 2.3. /dev/sr0

# fdisk -l

Disk /dev/vda: 10 GiB, 10737418240 bytes, 20971520 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
...

Disk /dev/sda: 25 GiB, 26843545600 bytes, 52428800 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

---

Inside the guest, the disks show up as a virtualized QEMU HARDDISK. The fact that it is a multipath disk is transparent to the guest. Only in the PowerKVM host, it is possible to see the paths as shown in Example 6-24 on page 184. If a path is offline and multipathing is configured correctly, the guest can still run without any interruption.

**Note:** In Example 6-26 on page 185, you can see two types of disks: /dev/vda and /dev/sda. /dev/vda uses the virtio driver and the QEMU HARDDISK uses a SCSI driver.

Instead of using `device='disk'` also `device='lun'` can be used. In this case, the disk/LUN is configured as a pass-through device and shows up with its storage origin, as shown in Example 6-28.

*Example 6-28 LUN mapped as path-through device*

---

```
[powerkvm-host]# cat mpio_disk.xml
<disk type='block' device='lun'>
  <driver name='qemu' type='raw' />
  <source dev='/dev/disk/by-id/dm-name-mpathb' />
  <target dev='sda' bus='scsi' />
</disk>

[linux-guest]# lsscsi
```

[0:0:0:0]	disk	IBM	2145	0000	/dev/sda
[0:0:0:2]	cd/dvd	QEMU	QEMU CD-ROM	2.3.	/dev/sr0

---

Kimchi also shows the added multipath disk as in the screen capture in Figure 6-9, but only when the device was added with `device='disk'` (not with `device='lun'`).

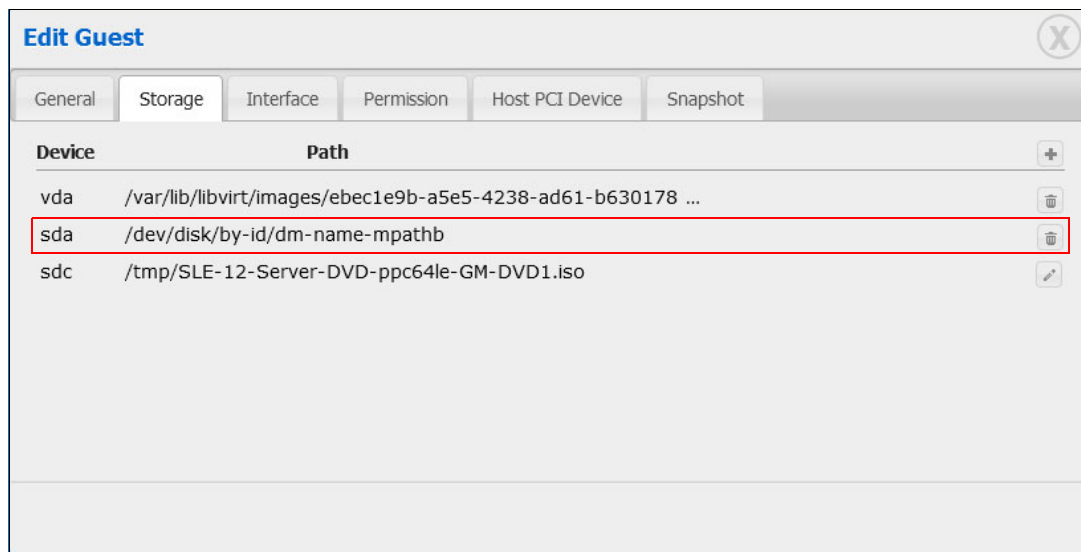


Figure 6-9 Added multipath disk in Kimchi

A multipath disk cannot be directly added to a guest using Kimchi. Therefore, it is preferred practice to include the disks into a storage pool as described in the next section, 6.6.3, “Multipath disks in a storage pool” on page 187.

### 6.6.3 Multipath disks in a storage pool

From a systems management point of view, the preferred practice is to use multipath disks inside a storage pool, which is discussed in this section. To create a storage pool using multipath disks, follow the steps as described in Example 4-6 on page 109 for an LVM2 block-based pool. After that, you can create a volume as shown in Example 4-11 on page 111 and attach it as described in Example 6-30 on page 189. For your convenience, Example 6-29 shows all steps in one example.

*Example 6-29 Using multipath disks in a storage pool*

---

```
[powerkvm-host]# virsh pool-define-as mp-pool logical --source-dev \
/dev/mapper/pkvm-pool1 --target /dev/mp-pool
Pool mp-pool defined
```

```
[powerkvm-host]# virsh pool-build mp-pool
Pool mp-pool built
```

```
[powerkvm-host]# virsh pool-start mp-pool
Pool mp-pool started
```

```
[powerkvm-host]# vgscan
Reading all physical volumes. This may take a while...
Found volume group "ibmpkvm_vg_data" using metadata type lvm2
Found volume group "ibmpkvm_vg_swap" using metadata type lvm2
```

```
Found volume group "ibmpkvm_vg_log" using metadata type lvm2
Found volume group "ibmpkvm_vg_root" using metadata type lvm2
Found volume group "mp-pool" using metadata type lvm2
```

```
[powerkvm-host]# virsh vol-create-as mp-pool myVol.qcow2 --format qcow2 10G
--allocation 4G
Vol myVol.qcow2 created
```

```
[powerkvm-host]# virsh vol-list mp-pool
```

Name	Path
------	------

myVol.qcow2	/dev/mp-pool/myVol.qcow2
-------------	--------------------------

```
[powerkvm-host]# virsh attach-disk linux-guest --source /dev/mp-pool/myVol.qcow2 \
--target sdc
Disk attached successfully
```

```
[linux-guest]# ls SCSI
[0:0:0:0] disk QEMU QEMU HARDDISK 2.3. /dev/sda
[0:0:0:2] cd/dvd QEMU QEMU CD-ROM 2.3. /dev/sr0
[0:0:0:3] disk QEMU QEMU HARDDISK 2.3. /dev/sdb
```

```
[linux-guest]# fdisk -l
```

```
Disk /dev/vda: 10 GiB, 10737418240 bytes, 20971520 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
...
```

```
Disk /dev/sda: 25 GiB, 26843545600 bytes, 52428800 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

```
Disk /dev/sdb: 10 GiB, 10737418240 bytes, 20971520 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

---

**Note:** All steps described in Example 6-29 on page 187 can be also achieved by using Kimchi as described in Chapter 3, “Managing hosts and guests from a Web interface” on page 61.

## 6.7 Hot plug

You can use a *hot plug* to add or remove devices on the system while it is online, without rebooting the system. This section covers SCSI disk hot plugs to the guest that is running under PowerKVM.

To add a disk to the guest online, use the **virsh attach-disk** command, as shown in Example 6-30.

*Example 6-30 vSCSI disk hot plug*

---

```
# virsh attach-disk sles11vm04 --source /var/lib/libvirt/images/sles11vm04_1.qcow2
--target sdf
Disk attached successfully
```

---

The disk hot plug works in the same way. A new LUN will be attached to an existing SCSI bus of a guest. If the guest has multiple SCSI adapters defined, libvirt picks the first one. If you want to use a specific one, use the **--address** argument.

After the disk is hot plugged to the guest, rescan the SCSI bus within the guest.

**Note:** You can use the **rescan-scsi-bus.sh** script from the **sg3-utils** package to rescan.

Example 6-31 shows the details of the process.

*Example 6-31 vSCSI hot plug attachment to the guest*

---

```
on the guest:
sles11vm04:~ # ls SCSI
[0:0:0:0] disk QEMU QEMU HARDDISK 1.6. /dev/sda
[0:0:0:1] cd/dvd QEMU QEMU CD-ROM 1.6. /dev/sr0

on the host:
[root@powerkvm ~]# virsh vol-create-as default hotplug.qcow2 --format qcow2 20G
--allocation 1G
Vol hotplug.qcow2 created

[root@powerkvm ~]# virsh attach-disk sles11vm04 --source
/var/lib/libvirt/images/hotplug.qcow2 --target sdc --persistent
Disk attached successfully

on the guest:
sles11vm04:~ # rescan-scsi-bus.sh
/usr/bin/rescan-scsi-bus.sh: line 647: [: 1.11: integer expression expected
Host adapter 0 (ibmvscsi) found.
Scanning SCSI subsystem for new devices
Scanning host 0 for SCSI target IDs 0 1 2 3 4 5 6 7, all LUNs
Scanning for device 0 0 0 0 ...
OLD: Host: scsi0 Channel: 00 Id: 00 Lun: 00
Vendor: QEMU Model: QEMU HARDDISK Rev: 1.6.
Type: Direct-Access ANSI SCSI revision: 05
Scanning for device 0 0 0 1 ...
OLD: Host: scsi0 Channel: 00 Id: 00 Lun: 01
Vendor: QEMU Model: QEMU CD-ROM Rev: 1.6.
Type: CD-ROM ANSI SCSI revision: 05
```

```

Scanning for device 0 0 0 2 ...
NEW: Host: scsi0 Channel: 00 Id: 00 Lun: 02
      Vendor: QEMU      Model: QEMU HARDDISK      Rev: 1.6.
      Type:   Direct-Access                      ANSI SCSI revision: 05
1 new or changed device(s) found.
0 device(s) removed.

```

```

sles11vm04:~ # lsscsi
[0:0:0:0]   disk      QEMU      QEMU HARDDISK      1.6.   /dev/sda
[0:0:0:1]   cd/dvd   QEMU      QEMU CD-ROM        1.6.   /dev/sr0
[0:0:0:2]   disk      QEMU      QEMU HARDDISK      1.6.   /dev/sdb

```

**Notes:**

--target SDC doesn't reflect the device name shown in the guest. Instead, it shows the name used by QEMU to identify the device. Also, based on the device name, the bus is automatically determined. Examples: sdX for vSCSI and vdX for Virtio devices, where X is a, b, c, and so on.

spapr-vscsi has a limit of seven devices being attached to the same bus.

To remove a disk from the system, use the **virsh detach-disk** command, as shown in Example 6-32.

*Example 6-32 Detach a disk from the guest*

**On the host:**

```

[root@powerkvm ~]# virsh detach-disk sles11vm04 sdc
Disk detached successfully

```

**On the guest:**

```

sles11vm04:~ # rescan-scsi-bus.sh -r
/usr/bin/rescan-scsi-bus.sh: line 647: [: 1.11: integer expression expected
Host adapter 0 (ibmvscsi) found.
Syncing file systems
Scanning SCSI subsystem for new devices and remove devices that have disappeared
Scanning host 0 for SCSI target IDs 0 1 2 3 4 5 6 7, all LUNs
Scanning for device 0 0 0 0 ...
OLD: Host: scsi0 Channel: 00 Id: 00 Lun: 00
      Vendor: QEMU      Model: QEMU HARDDISK      Rev: 1.6.
      Type:   Direct-Access                      ANSI SCSI revision: 05
Scanning for device 0 0 0 1 ...
OLD: Host: scsi0 Channel: 00 Id: 00 Lun: 01
      Vendor: QEMU      Model: QEMU CD-ROM        Rev: 1.6.
      Type:   CD-ROM                          ANSI SCSI revision: 05
sg2 changed: LU not available (PQual 3)
REM: Host: scsi0 Channel: 00 Id: 00 Lun: 02
DEL: Vendor: QEMU      Model: QEMU HARDDISK      Rev: 1.6.
      Type:   Direct-Access                      ANSI SCSI revision: 05
0 new or changed device(s) found.
1 device(s) removed.
sles11vm04:~ # lsscsi
[0:0:0:0]   disk      QEMU      QEMU HARDDISK      1.6.   /dev/sda
[0:0:0:1]   cd/dvd   QEMU      QEMU CD-ROM        1.6.   /dev/sr0

```

**Note:** To find the target name of a device to detach, first find the LUN number of the device (0:0:0:2 in this example). That is the third (starting from zero) letter from the alphabet. Therefore, in this example, that is **c**, so that is the SDC target to detach.

## 6.7.1 Adding a new vSCSI adapter

To add more than seven devices to a vSCSI bus, a new adapter needs to be created. To do so, a guest needs to be modified with the **virsh attach-device** command, as shown in Example 6-33.

*Example 6-33 Additional adapter definition*

---

```
# cat scsi.xml
  <controller type='scsi' index='2'>
    <address type='spapr-vio' reg='0x4000' />
  </controller>

# virsh attach-device sles11vm04 scsi.xml --config
Device attached successfully
```

---

**Note:** **--config** instructs libvirt to add an adapter definition, which is available after the next boot.

“Host PCI Device tab” on page 904.4.11, “CPU Hotplug” on page 127.







## Advanced topics

This chapter describes advanced topics that were not covered in the previous chapters. It also presents options to add your host to cloud environments.

After reading this chapter, you will have a deeper understanding of these PowerKVM-related topics:

- ▶ Install PowerKVM on a hardware Redundant Array of Independent Disks (RAID)
- ▶ Migrate guests to another host
- ▶ Add the host to a cloud environment
- ▶ Security
- ▶ PowerVC
- ▶ Docker usage

## 7.1 Install PowerKVM on a hardware RAID

Installing IBM PowerKVM V3.1.0 on a hardware RAID is a straightforward process. This section guides you through creating a RAID 10 array using the IBM Power RAID Configuration Utility, `iprconfig`. For more information about the `iprconfig` tool, see the following page:

<https://www.ibm.com/support/knowledgecenter/linuxonibm/liaau/liaau-iprutils.htm>

To proceed, you need to enter the Petitboot shell, as described in Figure 2-1 on page 33.

On the Petitboot shell, launch the `iprconfig` tool and you see the main window, as shown in Figure 7-1.

```

                                IBM Power RAID Configuration Utility

Select one of the following:

    1. Display hardware status
    2. Work with disk arrays
    3. Work with disk unit recovery
    4. Work with SCSI bus configuration
    5. Work with driver configuration
    6. Work with disk configuration
    7. Work with adapter configuration
    8. Download microcode
    9. Analyze log

Selection: █

e=Exit
```

Figure 7-1 `iprconfig` main window

Select option **2. Create a disk array**, as shown in Figure 7-2.

```
Work with Disk Arrays

Select one of the following:

1. Display disk array status
2. Create a disk array
3. Delete a disk array
4. Add a device to a disk array
5. Format device for RAID function
6. Format device for JBOD function
7. Work with hot spares
8. Work with asymmetric access
9. Force RAID Consistency Check
0. Migrate disk array protection

Selection: 2

e=Exit  q=Cancel
```

Figure 7-2 Create a disk array option

You are prompted to select the disk adapter, as shown in Figure 7-3. Select your disk adapter by pressing **1** and then **Enter**.

```
Create a Disk Array

Select the adapter.
Type choice, press Enter.
1=create a disk array

OPT Name  Resource Path/Address  Vendor  Product ID  Status
-----
1         FE               IBM     57D8001SISI0A  Operational

e=Exit  q=Cancel  t=Toggle
```

Figure 7-3 Select disk adapter

Next step is to select the disk units that will be part of the RAID. Select them by pressing **1** and **Enter**, as shown in Figure 7-4.

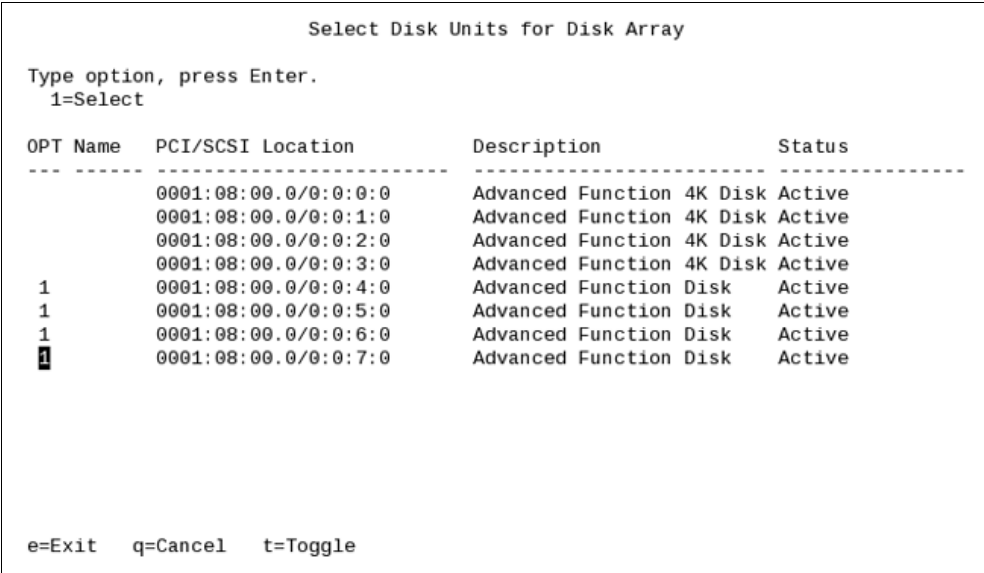


Figure 7-4 Select disk units

You are prompted to select the wanted RAID type, as shown in Figure 7-5. Press **c** to change the RAID type and then Enter to select. After that, press Enter to proceed.

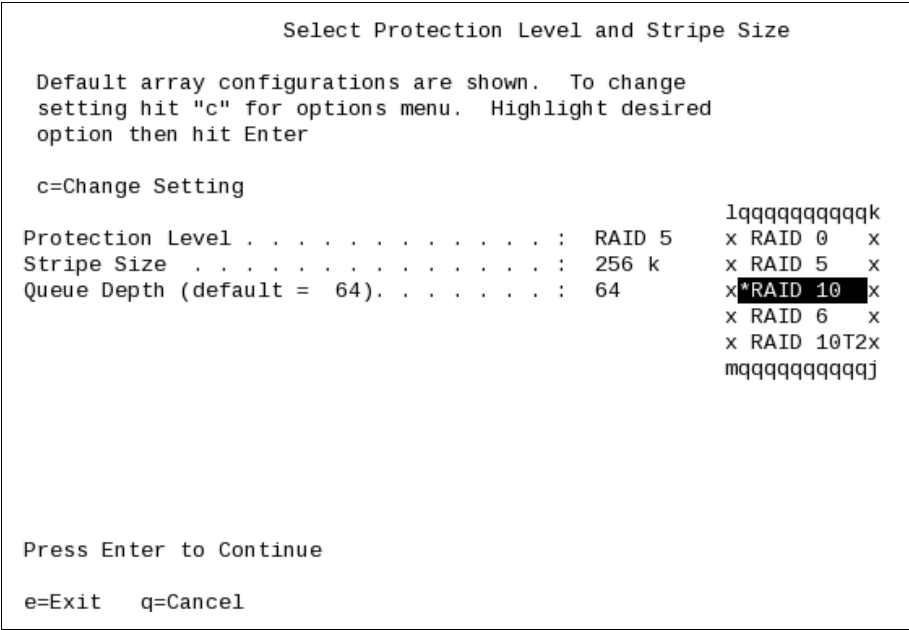


Figure 7-5 Select RAID type

A confirmation window is displayed, as shown in Figure 7-6.

Confirm Create Disk Array				
Press Enter to continue. q=Cancel to return and change your choice.				
OPT Name	Resource Path/Address	Vendor	Product ID	Status
1	FE	IBM	57D8001SISI0A	Operational
1	00-0C-04	IBM	HUC109060CSS600	Active
1	00-0C-05	IBM	HUC109060CSS600	Active
1	00-0C-06	IBM	HUC109060CSS600	Active
1	00-0C-07	IBM	HUC109060CSS600	Active
█				
q=Cancel t=Toggle				

Figure 7-6 Confirmation window for creating disk array

The message **Disk array successfully created** is displayed at the bottom of the window.

You can verify the status of your disk array by selecting option **1. Display disk array status** in the main window. You then see the status of the disk array, as shown in Figure 7-7.

Display Disk Array Status				
Type option, press Enter.				
1=Display hardware resource information details				
OPT Name	PCI/SCSI Location	Description	Status	
█ sda	0001:08:00.0/0:2:0:0	RAID 10 Disk Array	2% Rebuilt	
	0001:08:00.0/0:0:4:0	RAID 10 Array Member	Active	
	0001:08:00.0/0:0:5:0	RAID 10 Array Member	Active	
	0001:08:00.0/0:0:6:0	RAID 10 Array Member	Active	
	0001:08:00.0/0:0:7:0	RAID 10 Array Member	Active	
sdb	0003:04:00.0/1:2:0:0	RAID 10 Disk Array	Rebuilding	
	0003:04:00.0/1:0:10:0	RAID 10 Array Member	Remote	
	0003:04:00.0/1:0:11:0	RAID 10 Array Member	Remote	
	0003:04:00.0/1:0:12:0	RAID 10 Array Member	Remote	
	0003:04:00.0/1:0:9:0	RAID 10 Array Member	Remote	
e=Exit q=Cancel r=Refresh t=Toggle				

Figure 7-7 Disk array status

At this point, the `iprconfig` tool created a RAID 10 disk array. It still takes some hours until the disk array is fully built and ready to be used. After the disk array is ready, you can follow the installation instructions from section 2.1, “Host installation” on page 32.

## 7.2 Guest migration

IBM PowerKVM allows you to migrate guests between servers, reducing the downtime when moving host systems around data centers. Basically, there are three types of migration:

- ▶ **Offline:** After migration is complete, the migrating guest remains online on source host and shut down on destination host.
- ▶ **Online:** The migrating guest is paused in the source host and resumed in the destination host.
- ▶ **Live:** The migrating guest is copied without being shut down or paused. This type of migration takes longer and will sometimes not complete, depending on the workload.

To migrate a guest from one PowerKVM host to another, the following requirements must be satisfied:

- ▶ The Images volume is mounted at the same location in both hosts, usually `/var/lib/libvirt/images`.
- ▶ Source and destination hosts run the same PowerKVM version.
- ▶ Hosts have equal libvirt network configuration.
- ▶ Network traffic on TCP/IP ports 49152-49215 is allowed in both hosts. For migration over Secure Shell (SSH) protocol, make sure traffic on port 22 is also allowed to the destination host.

The `--persistent` option, used in the following examples, saves guest configuration on the destination host permanently. Otherwise, when this option is not specified, the guest configuration is erased from libvirt after the guest is shut down.

### 7.2.1 Offline migration

During offline migration, a guest definition is migrated without starting the guest on destination host and without stopping it on source host.

To perform an offline migration, you need to make sure both source and destination hosts share the same storage pool for guest disks, as for example:

- ▶ Fibre Channel
- ▶ iSCSI
- ▶ NFS

See section 4.2, “Managing storage pools” on page 107 for how to configure and use shared storage pools.

The switch `--offline` is specified in the `virsh migrate` command line options to indicate an offline migration.

The option `--undefinesource` is used to undefine the guest configuration on the source host. Otherwise, the guest will be configured on both servers after the migration is complete.

**Note:** Having a guest running on two different hosts can damage the guest disk image on the shared storage.

Example 7-1 on page 199 demonstrates how to perform an offline migration.

#### Example 7-1 Offline migration

---

```
root@source# virsh migrate --persistent --offline --undefinesource \  
  guest-name qemu+ssh://destination-host/system
```

---

## 7.2.2 Online migration

Online migration can be used when source and destination hosts do not share the same storage pool. In this case, the option `--copy-storage-all` is specified to copy the guest disk image to the destination host.

The online migration takes longer to complete than the offline one because the entire guest disk is copied over the network. The transfer time depends on the guest memory usage and network throughput between source and destination hosts.

To perform an online migration, the guest needs to be running on the source host. During the transfer, the guest appears as paused on the destination. After migration is complete, the guest is shut down on the source and resumed on the destination host.

**Note:** An attempt to perform an online migration with the guest shut down results in the following error message:

Error: Requested operation is not valid: domain is not running.

Example 7-2 shows how to perform an online migration.

#### Example 7-2 Online migration

---

```
root@source# virsh migrate --persistent --copy-storage-all \  
  guest-name qemu+ssh://destination-host/system
```

---

## 7.2.3 Live migration

In a live migration, the guest memory is transferred while it is still running on the source host. Remember that if the guest memory pages are changing faster than they are migrated, the migration can fail, time out, or not finish.

**Note:** Do not use Kimchi for guest XML disk setup if you plan to do live migration with SAN Fibre Channel.

Before proceeding, you need to make sure that the guest disk image is already available on the destination host. If you are not using a shared storage, make sure you perform an online migration first, as described in section 7.2.2, “Online migration” on page 199.

To perform a live migration, the guest must be running on the source host and must not be running on the destination host. During the migration, the guest is paused on the destination. After the transfer is complete, the guest is shut down on the source and then resumed on the destination host.

Example 7-3 shows how to perform a live migration.

*Example 7-3 Live migration*

---

```
root@source# virsh migrate --persistent --live \  
  guest-name qemu+ssh://destination-host/system
```

---

The `--timeout` option forces the guest to suspend when live migration exceeds the specified seconds, and then the migration completes offline.

Example 7-4 shows how to perform a live migration specifying a timeout of 120 seconds.

*Example 7-4 Live migration with timeout*

---

```
root@source# virsh migrate --persistent --live --timeout 120 \  
  guest-name qemu+ssh://destination-host/system
```

---

The migration can be interrupted due to an intense workload in the guest and can be started again with no damage to the guest disk image.

**Note:** Migration can fail if there is not enough contiguous memory space available in the target system.

## 7.3 Booting PowerKVM from Petitboot shell

Booting PowerKVM from a Petitboot shell can be used when you do not have a DHCP server to automatically provide the system with the boot configuration file, `pxe.conf`, as described in section 2.2.3, “Automated boot over DHCP” on page 46.

The only requirement is to have an HTTP server configured to serve `mlinuz`, `initrd.img`, `squashfs.img`, and packages repository. Refer to section “Configuration of an HTTP server” on page 46 for more details.

To boot PowerKVM from Petitboot shell, perform the following steps:

1. Place the `boot.sh` script, shown in Example 7-5 on page 201, in the `wwwroot` directory of your HTTP server so it can be downloaded from Petitboot shell.
2. Select **Exit to shell** in the Petitboot main menu to enter Petitboot shell prompt.
3. Download `boot.sh` script, for example:  

```
# wget http://server-address/boot.sh
```
4. Run `boot.sh`:  

```
# sh boot.sh
```

The script downloads the kernel and rootfs image, and hands execution over to the downloaded kernel image by calling the `kexec` command.



Example 7-5 is the `boot.sh` script that can be used to boot PowerKVM from Petitboot shell.

*Example 7-5 boot.sh*

---

```
#!/bin/sh
#
# Boot IBM PowerKVM V3.1.0 from Petitboot shell.

SERVER="http://server-address"

NIC="net0"
MAC="mac-address"
IP="ip-address"
GW="gateway"
NETMASK="netmask"
NS="dns-server"
HOSTNAME="your-system-hostname"

VMLINUZ="${SERVER}/ppc/ppc64le/vmlinuz"
INITRD="${SERVER}/ppc/ppc64le/initrd.img"
SQUASHFS="${SERVER}/LiveOS/squashfs.img"
REPO="${SERVER}/packages"
NET_PARAMS="ifname=${NIC}:${MAC} \
    ip=${IP}::${GW}:${NETMASK}:${HOSTNAME}:${NIC}:none nameserver=${NS}"
BOOT_PARAMS="rd.dm=0 rd.md=0 console=hvc0 console=tty0"

cd /tmp

wget $VMLINUZ
wget $INITRD

kexec -l vmlinuz --initrd initrd.img \
    --append="root=live:${SQUASHFS} repo=${REPO} ${NET_PARAMS} ${BOOT_PARAMS}"
kill -QUIT 1
```

---

Remember to update the following variables in the `boot.sh` sample script to meet your environment configuration:

- ▶ `SERVER`: The IP address or domain name of your HTTP server.
- ▶ `NIC`: The name of the network interface.
- ▶ `MAC`: The hardware address of the network interface.
- ▶ `IP`: The IP address of your host.
- ▶ `GW`: The gateway address.
- ▶ `NETMASK`: The network address.
- ▶ `NS`: The name server address.
- ▶ `HOSTNAME`: The host name of your host system.

## 7.4 Security

This section gives you an overview of some security aspects present in IBM PowerKVM V3.1.0.

## 7.4.1 SELinux

SELinux is a Linux kernel security module that provides mandatory access control (MAC). One of its features is that it does not allow programs to access information of other programs running in different contexts. That provides guest isolation from the rest of the other host applications. No other program on the host can affect guest functioning without being explicitly allowed by SELinux rules, if SELinux is running in Enforcing mode.

By default, the PowerKVM Live DVD and the target system run in Enforcing mode. You can verify the SELinux policy by running the following command:

```
# getenforce
Enforcing
```

You can change the runtime policy to Permissive by running the following command:

```
# setenforce Permissive
```

Then, the **getenforce** command shows the new policy:

```
# getenforce
Permissive
```

The policy can be updated permanently by changing the content of the `/etc/selinux/config` file. Example 7-6 is a sample SELinux configuration file with Enforcing policy.

*Example 7-6 Sample SELinux configuration file*

---

```
# /etc/selinux/config
#
# SELinux configuration file.
```

```
SELINUX=enforcing
SELINUXTYPE=targeted
```

---

Changes in the `/etc/selinux/config` file only take place after reboot.

The SELinux context of a file can be updated by using the **chcon** command. The following example updates the context of `guest69-disk.img` file to `samba_share_t` type:

```
# chcon -t samba_share_t guest69-disk.img
```

The default SELinux context for the guest disk files under `/var/lib/libvirt/images` is `virt_image_t`. Run the **restorecon** command to restore the original SELinux context of a file. For example:

```
# restorecon -v /var/lib/libvirt/images/guest69-disk.img
restorecon reset /var/lib/libvirt/images/guest69-disk.img context \
    system_u:object_r:samba_share_t:s0->system_u:object_r:virt_image_t:s0
```

The **restorecon** command reads the files in the `/etc/selinux/targeted/contexts/files/` directory, to see which SELinux context files it should have.

For more details about SELinux, visit the SELinux Project page at:

<http://www.selinuxproject.org>

## 7.4.2 System updates

On PowerKVM, you can keep your system up to date by using **yum update** or **ibm-update-system** commands. Both of them can download and install RPM packages from the following external repository:

<http://public.dhe.ibm.com/software/server/POWER/Linux/powerkvm/release/3.1.0/updates>

Example 7-7 shows the content of the default repository configuration file.

*Example 7-7 Default repository configuration file*

---

```
# /etc/yum.repos.d/base.repo
#
# PowerKVM repository configuration file

[powerkvm-updates]
name=IBM PowerKVM $ibmver - $basearch
baseurl=http://public.dhe.ibm.com/software/server/POWER/Linux/powerkvm/$ibmmilestone
ne/$ibmver/updates
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-ibm_powerkvm-$ibmver-$ibmmilestone
skip_if_unavailable=1

[powerkvm-debuginfo]
name=IBM PowerKVM Debuginfo - $ibmver - $basearch
baseurl=http://public.dhe.ibm.com/software/server/POWER/Linux/powerkvm/$ibmmilestone
ne/$ibmver/debuginfo
enabled=0
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-ibm_powerkvm-$ibmver-$ibmmilestone
skip_if_unavailable=1
```

---

One preferred practice is to use the **ibm-update-system** command to apply system updates because it installs the recommended packages that were not installed by default at installation time.

For example, when a package is added to the PowerKVM image in a further release, you can still get it installed on your system next time you run the **ibm-update-system** tool.

Another preferred practice is *not* to use external packages repositories. Otherwise, you can damage your system when installing software from non-trusted sources.

With the **ibm-update-system** command, you can also update your system using a local PowerKVM ISO image. The following example shows how to apply updates using a local image:

```
# ibm-update-system -i ibm-powerkvm.iso
```

When the command is entered, you are prompted to answer if you want to proceed with the update. The **-y** option can be used to assume yes and skip this question.

Run **ibm-update-system --help** to obtain more information about the supported options.

## 7.5 Cloud management

PowerKVM hosts can be managed just like another KVM host. PowerKVM includes Kimchi in its software stack to provide a friendly user interface for platform management in a single server. For larger management, such as cloud environments, you can use IBM PowerVC or IBM Cloud Manager with OpenStack. Table 7-1 shows the possible management options for PowerKVM.

Table 7-1 Virtualization management systems

Management software	Capabilities	Installation type
Kimchi	Single server	Installed on the host
PowerVC	Multiple servers	Requires a dedicated server
IBM Cloud Manager with OpenStack	Multiple servers	Requires a dedicated server
OpenStack Nova Controller Services	Multiple servers	Installed on the host or dedicated server

Kimchi is an open source project for virtualization management within one server. IBM PowerVC and IBM Cloud Manager are advanced management solutions created and maintained by IBM, built on OpenStack.

*OpenStack* is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a data center. OpenStack Compute (the Nova component) has an abstraction layer for compute drivers to support different hypervisors, including QEMU or KVM, through the libvirt virtualization API.

The following sections introduce the virtualization management systems that can be used to manage the PowerKVM servers in cloud environments.

### 7.5.1 IBM PowerVC

IBM Power Virtualization Center (IBM PowerVC) is an advanced virtualization manager for creating and managing virtual machines on IBM Power Systems servers by using the PowerKVM hypervisor. PowerVC simplifies the management of virtual resources in your Power Systems environment. It is built on OpenStack technology to deliver an infrastructure as a service (IaaS) within the cloud. With PowerVC, you can deploy your VMs among other tasks, as shown in the following list:

- ▶ Create virtual machines and then resize and attach volumes to them.
- ▶ Import existing virtual machines and volumes so they can be managed by IBM PowerVC.
- ▶ Monitor the use of the resources that are in your environment.
- ▶ Migrate virtual machines while they are running (hot migration).
- ▶ Deploy images quickly to create new virtual machines that meet the demands of your ever-changing business needs.

This section gives an overview of how you can add PowerKVM hosts as a compute node and deploy cloud images by using PowerVC. For more detailed information about how to install and configure PowerVC, see *IBM PowerVC Version 1.2.3: Introduction and Configuration*, SG24-8199.

You can install PowerVC on a separate host and control your compute nodes through the interface by using your web browser. Figure 7-8 shows the PowerVC management interface.

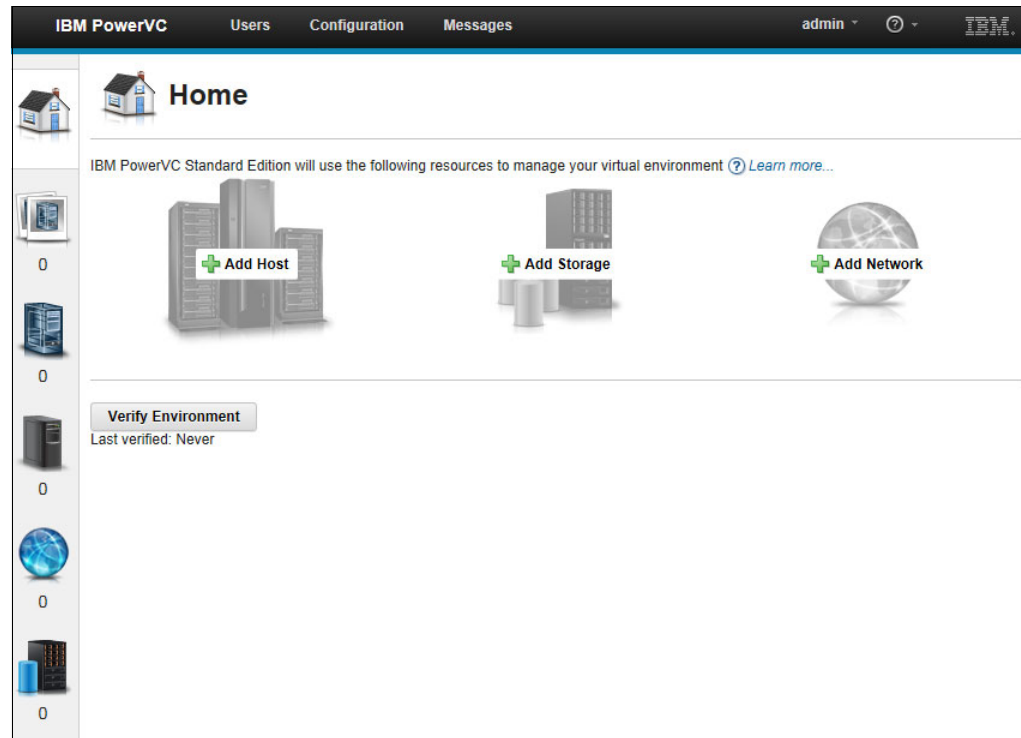


Figure 7-8 PowerVC interface for advanced virtualization management

To connect to a PowerKVM host, simply enter the credentials for the host as shown in Figure 7-9. PowerVC automatically installs the necessary OpenStack modules on the PowerKVM host and adds the host as a compute node in PowerVC.

### Add Host

Specify the details for the PowerKVM host you want to add.

\* Host name or IP address:

\* Display name: [?](#)

\* Storage directory: [?](#)

\* User ID:

Authentication type:  
☒ Password ☐ SSH key

\* Password:

[Show the mounted file systems that are on this host.](#)

Add Host

Cancel

Figure 7-9 Adding a new host to PowerVC

The new added host is listed on the *Hosts* panel, as shown in Figure 7-10.

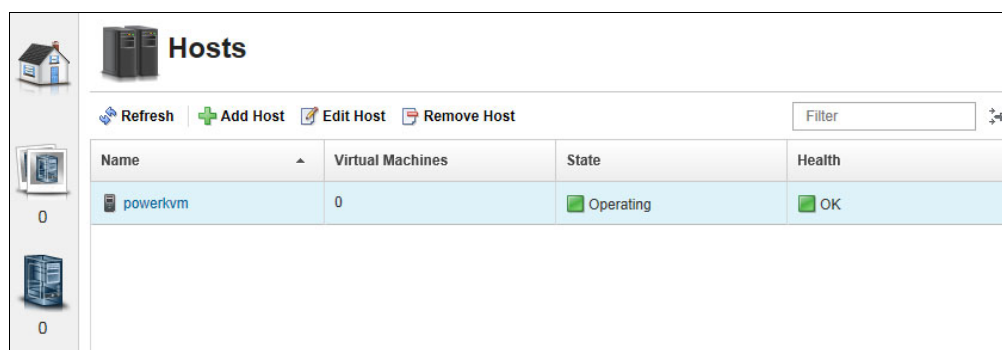


Figure 7-10 Hosts list on PowerVC

Before importing images and creating instances on PowerVC, configure the network and storage settings.

When using PowerVC together with PowerKVM, the networking is done using an Open vSwitch environment, not the standard bridging that is commonly used with PowerKVM. For a simple environment, PowerVC prepares the Open vSwitch environment when connecting to a new PowerKVM host that does not have an open vSwitch environment configured. Example 7-8 shows a simple vSwitch environment configured by PowerVC.

Example 7-8 Simple open vSwitch environment configured by PowerVC

---

```
# ovs-vsctl show
0b140048-c1e2-428e-a529-516a347f283c
    Bridge default
        Port "enP3p9s0f0"
            Interface "enP3p9s0f0"
        Port default
            Interface default
                type: internal
        Port phy-default
            Interface phy-default
                type: patch
                options: {peer=int-default}
    Bridge br-int
        fail_mode: secure
        Port br-int
            Interface br-int
                type: internal
        Port int-default
            Interface int-default
                type: patch
                options: {peer=phy-default}
    ovs_version: "2.0.0"
```

---

PowerVC supports deploy and capture features with local or NFS storage. Before deploying a virtual machine on PowerKVM hosts, you must upload an image first, as shown in Figure 7-11. You can upload ISO or QCOW2 image types. The supported operating systems are Red Hat Enterprise Linux (RHEL), SUSE Linux Enterprise Server (SLES), and Ubuntu. The distribution can be Little Endian or Big Endian because both architectures are supported by PowerKVM running on a POWER8 system. You can use the boot.iso image provided in the installation DVD as a minimal image suitable for installation over a network.

**Upload Image**  
Specify the details for the image that you want to upload. ? [Learn more...](#)

\* Image name:

\* Operating system:

\* Image type:

\* Image file:

Figure 7-11 Image upload on PowerVC

It is also possible to use a virtual machine as a *golden image* that contains the operating system, perhaps some applications, and also other customization. This virtual machine can be equipped with an activation mechanism, such as cloud-init or the IBM Virtual Solutions Activation Engine (VSAE) that changes settings like an IP address, host name, or the SSH keys when deploying the image to a new virtual machine. A virtual machine that contains the golden image can be captured and then used for deployments.

Figure 7-12 shows images that are ready to be deployed. The figure contains ISO boot images as well as prepared so-called golden images (snapshots).

Images

Refresh

Deploy

Delete

Upload

Edit Description

Name	State	Operating System	Type
RHEL7.1	Active	rhel	Image
RHEL71_Goldenimage	Active	rhel	Snapshot
SLES12	Active	sles	Image
SLES12_Goldenimage	Active	sles	Snapshot
Ubuntu14.10	Active	ubuntu	Image
Ubuntu14_10_Goldenimage	Active	ubuntu	Snapshot

Figure 7-12 Displaying images

To deploy an image, simply use the **Deploy** button as shown in Figure 7-12 and enter the required data for the new guest as shown in Figure 7-13.

In the deploy dialog, all necessary data is gathered to install and configure the new guest on the PowerKVM host. For the size of the new guest, so-called *compute templates* are used. In native OpenStack, compute templates are also referred to as *flavors*. A compute template defines the number of virtual processors, sockets, cores and threads, the memory size, and the size for the boot disk.

Images

Deploy SLES12\_Goldenimage

Deploy SLES12\_Goldenimage

General

Virtual machine name:

linux-guest

Instances:

1

Deploy target:

Default Grou...

Compute template:

VM\_small

Processors

2

Memory (MB)

1,024

Disk size (GB)

10

Custom vCPU Distribution

Sockets: 1 Cores: 1 Threads: 2

Current and projected use:

Your projected usage based on your selections is shown in this color.

Virtual machines (2)

Processors (6 )

13%

48 Total

Memory (3,584 MB)

3%

129,750 MB Total

Disk size (38 GB)

17%

222 GB Total

Network

Primary network (system default gateway):

net-10-150-60

Specify IP address:

Picked from IP pool

Activation Input

Deploy

Cancel

Figure 7-13 Deploy an image on PowerVC

208

IBM PowerKVM: Configuration and Use



For boot.iso deployments, you can connect to your deployed images by using the **virsh console** command. For images based on preinstalled virtual machines with the network configured, you can connect using SSH.

For attachment of further data disks, also iSCSI volumes can be attached. The creation of a disk is usually done in PowerVC, but also existing volumes can be imported. Figure 7-14 shows the attachment of a new data disk using iSCSI, connected to an IBM Storwize V7000 system.

**Attach Volume**  
Select an existing volume or create a new volume to attach to the selected virtual machine.

☐ Attach an existing volume to this virtual machine.

☒ Attach a new volume to this virtual machine.

Create a new volume to attach to this virtual machine.

\* Storage template: tce-v7000-iscsi base template [? Learn about storage templates](#)

\* Volume name: linux-guest-data

Description: Data LUN for linux-guest

\* Size (GB): 100

Real Size: 2 GB

**Current Storage Used**

Storage (419.84 GB) 5,580.8 GB Total

8%

The projected storage use based on the selected volume size is shown in this color.

Storage Provider: tce-v7000-iscsi  
Volume Type: Thin Provisioned  
Storage Pool: tce-datapool  
Available Capacity: 5,160.96 GB  
Real Capacity: 2% of virtual capacity

**Attach** **Cancel**

Figure 7-14 Attachment of an iSCSI disk in PowerVC

PowerVC provides additional advanced functions, such as:

- ▶ Grouping of hosts
- ▶ Placement policies to select the best destination for a guest, depending on memory or CPU allocation, CPU utilization, or simple rules such as striping (round robin) or packaging (a host is filled with guests to a certain amount, before the next one is used).
- ▶ Collocation rules
  - Affinity rules define that several guests must reside on the same host, for instance for performance reasons.
  - Anti-affinity rules define that several guests must reside on different hosts, for instance for availability reasons.
- ▶ Dynamic Resource Optimizer (DRO). DRO monitors the resource usage of physical hosts and virtual machines in a cloud environment. When a host becomes overused, DRO can migrate virtual machines to hosts that have less utilization.
- ▶ Customization scripts for custom configuration of a guest.
- ▶ IP address pool: IP addresses can be either defined in the deploy dialog or can be autoselected from a pool.

- ▶ Live migration of a guest to another host.
- ▶ Remote restart: Restarting a guest on another host, when the source host is down or in a defect state.
- ▶ Maintenance mode: Live migration of all guests in order to shut down a host for maintenance.

For implementing PowerVC connecting to PowerKVM servers, an IBM Systems Lab Services Techbook can be found here:

<https://ibm.box.com/PowerVC123-on-PowerKVM>

See also *IBM PowerVC Version 1.2.3: Introduction and Configuration*, SG24-8199 for more information about how to manage PowerKVM hosts using PowerVC.

## 7.5.2 IBM Cloud Manager with OpenStack

IBM Cloud Manager with OpenStack, formerly offered as IBM SmartCloud® Entry, can be used to get started with private clouds that can scale users and workloads. IBM Cloud Manager can be also used to attach to a public cloud, such as IBM SoftLayer®. It is based on the OpenStack project and provides advanced resource management with a simplified cloud administration and full access to OpenStack APIs.

These are among the benefits of using IBM Cloud Manager with OpenStack for Power:

- ▶ Full access to OpenStack APIs
- ▶ Simplified cloud management interface
- ▶ All IBM server architectures and major hypervisors are supported. This includes x86 KVM, KVM for IBM z™ Systems, PowerKVM, PowerVM, Hyper-V, IBM z/VM®, and VMware.
- ▶ Chef installation enables flexibility to choose which OpenStack capabilities to use
- ▶ AutoScale using the OpenStack Heat service
- ▶ Manage Docker container services

IBM Cloud Manager comes with two graphical user interfaces: The IBM Cloud Manager Dashboard, which is in OpenStack also referred to as *Horizon* and the IBM Cloud Manager Self Service portal, which is only available with the IBM Cloud Manager with OpenStack product. Figure 7-15 shows a screen capture of the new IBM Self Service portal that is shipped with IBM Cloud Manager Version 4.3.

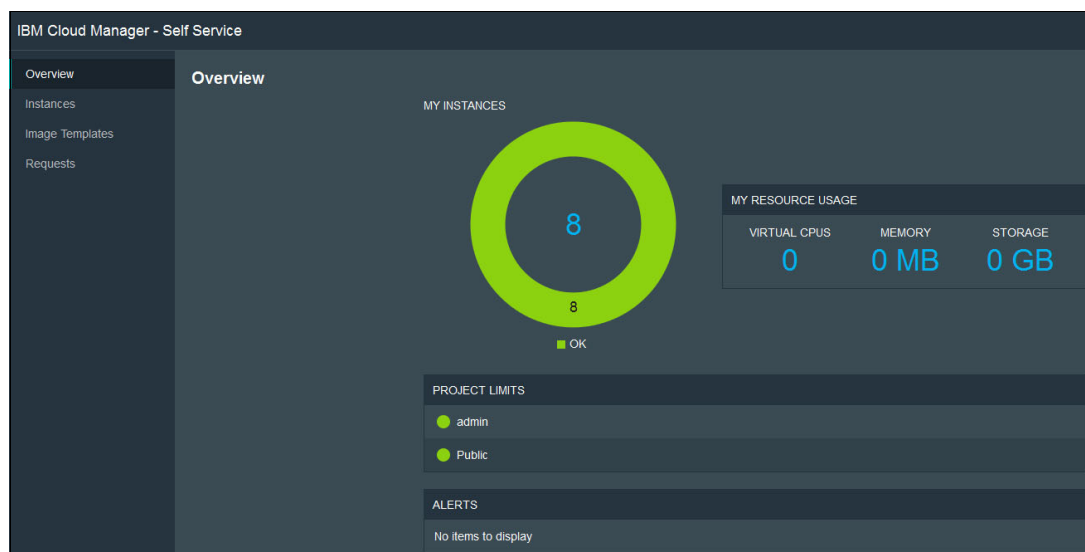


Figure 7-15 IBM Cloud Manager with OpenStack Self Service portal

For more information about the IBM Cloud Manager with OpenStack, refer to the documentation that can be found here:

<http://ibm.co/1cc5r7o>

### 7.5.3 OpenStack controller services

The PowerKVM host can be managed by the open source controller services that are maintained by the OpenStack community. The compute and controller services on OpenStack enable you to launch virtual machine instances.

This section gives an overview of how to install and configure compute controller services to add your PowerKVM server to OpenStack. You can configure these services on a separate node or the same node. A dedicated compute node requires only *openstack-nova-compute*, the service that launches the virtual machines on the PowerKVM host.

RPM is the package management system used by PowerKVM. To install the open source version of OpenStack compute services on PowerKVM, get the RPM packages from your preferred Linux distribution or build your own packages.

IBM PowerKVM does not bundle OpenStack packages. The installation instructions in this section are based on Fedora repositories:

<http://repos.fedorapeople.org/repos/openstack>

The link has several subdirectories for the OpenStack releases, especially the Liberty release, which was the latest at the time of writing.

See the online documentation for how to install and configure compute controller services on OpenStack. You can find detailed information about how to install and configure OpenStack services on Red Hat Enterprise Linux, CentOS, SUSE Linux Enterprise Server, and Ubuntu on the OpenStack.org website:

<http://docs.openstack.org/#docs-main-body>

**Note:** IBM PowerKVM version 3.1.0 does not include OpenStack community packages. You can choose to install IBM Cloud Manager or IBM PowerVC to have full integration and support for cloud services.

## Compute node

Follow these steps to add a PowerKVM host as a compute node to an existing cloud controller:

1. Install the *openstack-nova-compute* service. These dependencies are required:
  - openstack-nova-api
  - openstack-nova-cert
  - openstack-nova-conductor
  - openstack-nova-console
  - openstack-nova-novncproxy
  - openstack-nova-scheduler
  - python-novaclient
2. Edit the `/etc/nova/nova.conf` configuration file:
  - a. Set the authentication and database settings.
  - b. Configure the compute service to use the RabbitMQ message broker.
  - c. Configure Compute to provide remote console access to instances.
3. Start the Compute service and configure it to start when the system boots.
4. Confirm that the compute node is listed as a host on nova, as shown in Example 7-9.

**Tip:** Export Nova credentials and access the API from any system that can reach the controller machine.

Example 7-9 Listing hosts on Nova

```
[user@laptop ~(keystone_admin)]$ nova host-list
```

host_name	service	zone
controller	consoleauth	internal
controller	scheduler	internal
controller	conductor	internal
controller	network	internal
controller	cert	internal
powerkvm	network	internal
powerkvm	compute	nova

## Controller node

As a preferred practice, choose a separate server for the controller node when planning to scale horizontally. The instructions to install the controller are in the OpenStack online documentation. There are no special requirements. A minimum OpenStack setup requires the authentication, image, and network services in addition to the controller services. These services include keystone, glance, and neutron (or nova-network).

To configure the compute services in the controller node, install these packages:

- ▶ openstack-nova-api
- ▶ openstack-nova-cert
- ▶ openstack-nova-conductor
- ▶ openstack-nova-console
- ▶ openstack-nova-novncproxy
- ▶ openstack-nova-scheduler
- ▶ python-novaclient

**Note:** It is also possible to add a PowerKVM compute node to an existing cloud controller running on an IBM x86 server. You might use host aggregates or an availability zone to partition mixed architectures into logical groups that share specific types or images.

After the services are running in the controller, you can deploy your images on the PowerKVM host. To deploy an image and specify the host that you want to run, use the **--availability-zone** option, as shown in Example 7-10.

*Example 7-10 Deploying an image using a nova command line*

```
[user@laptop ~(keystone_admin)]$ nova boot --image my-image --flavor 3 --availability-zone nova:powerkvm vm054
```

Property	Value
OS-DCF:diskConfig	MANUAL
OS-EXT-AZ:availability_zone	nova
OS-EXT-SRV-ATTR:host	-
OS-EXT-SRV-ATTR:hypervisor_hostname	-
OS-EXT-SRV-ATTR:instance_name	instance-00002de6
OS-EXT-STS:power_state	0
OS-EXT-STS:task_state	scheduling
OS-EXT-STS:vm_state	building
OS-SRV-USG:launched_at	-
OS-SRV-USG:terminated_at	-
accessIPv4	
accessIPv6	
adminPass	PcMpnee22G7M
config_drive	
created	2014-05-28T19:25:18Z
flavor	m1.medium (3)
hostId	
id	01271354-59dc-480f-a2db-53682fc3d37e
image	my-image (36b70fda-497d-45ff-899a-6de2a3616b32)
key_name	-
metadata	{}
name	vm054
os-extended-volumes:volumes_attached	[]
progress	0
security_groups	default
status	BUILD
tenant_id	ad13d65a473f4cafa1db9479c3f7c645

updated	2014-05-28T19:25:18Z
user_id	9b2ee46e7bbc405bb1816603445de08c
+-----+-----+	

Besides PowerKVM, there are several hypervisors that are supported. For details, see the HypervisorSupportMatrix page on the OpenStack website:

<https://wiki.openstack.org/wiki/HypervisorSupportMatrix>

For more detailed options for creating virtual machines on OpenStack, see the online documentation.

## 7.6 Docker usage

This section covers the basic usage of Docker based on the existing development packages. It is not expected to have any difference from the released Docker packages for PowerKVM.

As explained in the 1.4, “Docker” on page 22, Docker provides an infrastructure for containers, aiming to build and run distributed applications. Even though Docker is focused on application containers, this section uses it as a system container.

### 7.6.1 Docker installation

To install Docker, you need to add the Development Kit repository in the PowerKVM host as described in 8.2, “Installation” on page 229. When you have the Development Kit repository added on the host, you should install Docker using the yum package manager, as explained in Example 7-11.

*Example 7-11 Installing Docker package on PowerKVM*

```
# yum install docker
...
--> Running transaction check
---> Package docker.ppc64le 1:1.7.0-22.gitdcff4e1.5.el7_1.2 will be installed
--> Processing Dependency: docker-selinux >= 1:1.7.0-22.gitdcff4e1.5.el7_1.2 for
package: 1:docker-1.7.0-22.gitdcff4e1.5.el7_1.2.ppc64le
--> Processing Dependency: uberchain-ppc64le for package:
1:docker-1.7.0-22.gitdcff4e1.5.el7_1.2.ppc64le
--> Running transaction check
---> Package docker-selinux.ppc64le 1:1.7.0-22.gitdcff4e1.5.el7_1.2 will be
installed
---> Package uberchain-ppc64le.ppc64le 0:8.0-4.pkvm3_1_0 will be installed
--> Finished Dependency Resolution
```

Dependencies Resolved

```
=====
Package Arch Version
Repository Size
=====
Installing:
```

```

docker                                ppc64le
1:1.7.0-22.gitdcff4e1.5.e17_1.2      powerkvm-iso      4.1
M
Installing for dependencies:
docker-selinux                        ppc64le
1:1.7.0-22.gitdcff4e1.5.e17_1.2      powerkvm-iso      48
k
uberchain-ppc64le                    ppc64le           8.0-4.pkvm3_1_0
powerkvm-iso                          72 M

Transaction Summary
=====
=====
Install 1 Package (+2 Dependent packages)

Total download size: 76 M
Installed size: 400 M
Is this ok [y/d/N]: y
Downloading packages:
-----
-----
Total
293 MB/s | 76 MB 00:00:00
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
Installing:1:docker-selinux-1.7.0-22.gitdcff4e1.5.e17_1.2.ppc64le
1/3

Installing : uberchain-ppc64le-8.0-4.pkvm3_1_0.ppc64le
2/3
Installing : 1:docker-1.7.0-22.gitdcff4e1.5.e17_1.2.ppc64le
3/3
Verifying : uberchain-ppc64le-8.0-4.pkvm3_1_0.ppc64le
1/3
Verifying : 1:docker-1.7.0-22.gitdcff4e1.5.e17_1.2.ppc64le
2/3
Verifying : 1:docker-selinux-1.7.0-22.gitdcff4e1.5.e17_1.2.ppc64le
3/3

Installed:
docker.ppc64le 1:1.7.0-22.gitdcff4e1.5.e17_1.2

Dependency Installed:
docker-selinux.ppc64le 1:1.7.0-22.gitdcff4e1.5.e17_1.2
uberchain-ppc64le.ppc64le 0:8.0-4.pkvm3_1_0

Complete!

```

---

To verify if your Docker installation is fine and working, you can run the **docker info** command as shown in Example 7-12.

*Example 7-12 docker info output*

---

```
# docker info
Containers: 0
Images: 0
Storage Driver: devicemapper
  Pool Name: docker-253:1-134273-pool
  Pool Blocksize: 65.54 kB
  Backing Filesystem: extfs
  Data file: /dev/loop1
  Metadata file: /dev/loop2
  Data Space Used: 307.2 MB
  Data Space Total: 107.4 GB
  Data Space Available: 10.01 GB
  Metadata Space Used: 733.2 kB
  Metadata Space Total: 2.147 GB
  Metadata Space Available: 2.147 GB
  Udev Sync Supported: true
  Deferred Removal Enabled: false
  Data loop file: /var/lib/docker/devicemapper/devicemapper/data
  Metadata loop file: /var/lib/docker/devicemapper/devicemapper/metadata
  Library Version: 1.02.93-RHEL7 (2015-01-28)
Execution Driver: native-0.2
Logging Driver: json-file
Kernel Version: 3.18.22-359.el7_1.pkvm3_1_0.4000.1.ppc64le
Operating System: IBM_PowerKVM 3.1.0
CPUs: 8
Total Memory: 126.975 GiB
Name: localhost
ID: ES2X:KFCA:YLIE:3TC4:H7Y6:YHF4:WFSD:XYFH:RCXN:K35K:G4KM:I2WL
```

---

## Docker errors

If you run the command shown in Example 7-12, and the output was an error, for example, an output similar to Example 7-13, it means that you do not have access to the running Docker service. It might be because your user does not have privileges to access the Docker service. The other option is that the Docker service is down. In that case, you can start the service with:

```
# systemctl start docker
```

*Example 7-13 Docker service access error*

---

```
Get http://var/run/docker.sock/v1.20/info: dial unix /var/run/docker.sock: no
such file or directory. Are you trying to connect to a TLS-enabled daemon without
TLS?
```

---



On the other side, if you run Docker and it shows an output like Example 7-14, it means that the golang libraries were not found. Assure that the golang libraries are installed and visible. The golang libraries are usually at /opt/ibm/lib64, and two file system links as follows should solve this problem:

```
# ln -s /opt/ibm/lib64/libgo.so /lib64/libgo.so
# ln -s /opt/ibm/lib64/libgo.so.7 /lib64/libgo.so.7
```

*Example 7-14 Docker failing due to shared library not found*

---

```
$ docker
docker: error while loading shared libraries: libgo.so.7: cannot open shared
object file: No such file or directory
```

---

## 7.6.2 Image management

As described in section 1.4.2, “Docker hub” on page 26, Docker is strictly connected to the Docker hub image database service. The easiest way to start playing with Docker on PowerKVM is by using images available at Docker hub.

You can search for ready-to-deploy container images by using the **docker search** command, which lists all the images that contain a specified word in the images name available at Docker hub.

Example 7-15 shows part of the output **docker search** command searching for **ppc64** images. By convention, POWER images have ppc64 or ppc64le in its name but it is important to notice that the image uploader is responsible to name the image, so an image can have ppc64 in its name but not necessarily designed to run on a POWER architecture.

*Example 7-15 Searching for images*

---

```
$ docker search ppc64
```

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
ppc64le/busybox-ppc64le				0
ppc64le/debian				0
ppc64le/ubuntu				0
ppc64le/buildpack-deps				0
ppc64le/busybox				0
ppc64le/gcc				0
ppc64le/hello-world				0
ibmcom/gccgo-ppc64le				0
ibmcom/ubuntu-ppc64le				0
ibmcom/busybox-ppc64le				0
ibmcom/unshare-ppc64le				0
ibmcom/hello-world-ppc64le				0

---

After selecting the image to download, use the **pull** command to grab the image from the remote archive and put it inside your Docker directory (Example 7-16). The container images are saved in an AUFS (Another Union File System) format at `/var/lib/dokcer/aufs`.

Example 7-16 Downloading a remote Docker image

```
$ docker pull ppc64le/debian
latest: Pulling from ppc64le/debian
8a78fb91edd3: Pull complete
Digest: sha256:17c58857e4321300c3ac516fdd9eeb4de017a15a9c69b9b3fbdd7115287343a4
Status: Downloaded newer image for ppc64le/debian:latest
```

Docker images can be listed by using the **docker images** command, as shown in Example 7-17.

Example 7-17 Listing container images

```
$ sudo docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
ppc64le/busybox	latest	a1c0a636f835	46 hours ago	4.871 MB
ppc64le/debian	latest	ba92052e52c6	47 hours ago	127.6 MB

**Note:** You can also see most of the Docker images for POWER at the Docker hub web at the following address:

<https://hub.docker.com/u/ppc64le>

## 7.6.3 Container management

Based on a given container image, you can create a container using two different ways. You can either use the **run** or **create** Docker commands. Both of these options are covered in this section.

Starting a container by using the **run** command is more straightforward because only one command is enough to create the container, start it, and have a shell access to it. Using the **create** command requires three different commands that are shown in “Creating a container” on page 220.

**Note:** Your user/login needs to have proper access to Docker in order to access Docker commands. Otherwise, Docker complains by using the following message:

```
time="2015-11-16T09:29:45-05:00" level=fatal msg="Get
http://var/run/docker.sock/v1.18/containers/json: dial unix
/var/run/docker.sock: permission denied. Are you trying to connect to a
TLS-enabled daemon without TLS?"
```

### Running a command on an image

After you have an image on your Docker environment, you can run a command inside the Docker image easily. In this case, it starts a container with the image you specified, starts the container, and runs the command inside the container.

For example, if you want to have a bash shell command inside the container, you can use the Docker command **run**, using the following arguments:

```
$ docker run -t -i <image> <command>
```

Where *-i* means you want to use the container interactively and *-t* asks Docker engine to create a terminal for you.

Example 7-18 shows a Debian container being started using the *ppc64le/debian* image downloaded previously. In this example, the bash shell is started and returned to the user in an interactive terminal. After that, you will be inside the container, and any file you see or execute will come from the container file system and will be executed within the container context.

Example 7-18 Starting a container

```
$ docker run --name itso -t -i ppc64le/debian /bin/bash
```

```
root@e022e2a9cb66:/# cat /etc/os-release
PRETTY_NAME="Debian GNU/Linux 8 (jessie)"
NAME="Debian GNU/Linux"
VERSION_ID="8"
VERSION="8 (jessie)"
ID=debian
HOME_URL="http://www.debian.org/"
SUPPORT_URL="http://www.debian.org/support/"
BUG_REPORT_URL="https://bugs.debian.org/"
```

After you leave the shell, the default container dies because Docker was created to be an application container. To see the container that is already down, you can use the **ps** Docker command. No argument is necessary. You only list the active containers, as shown in Example 7-19. You can also keep the container running even after the shell dies. In order to do that, you must use the argument *--restart*.

Example 7-19 Listing active containers

\$ docker ps	CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
	5c1da35da075	ppc64le/debian:latest	"/bin/bash"	14 seconds ago	Up 13 seconds		itso

On the other side, if you want to see all the containers, either in running or suspended states, you can use the *-a* flag, as shown in Example 7-20.

Example 7-20 Listing all containers

\$ docker ps -a	CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
	5c1da35da075	ppc64le/debian:latest	"/bin/bash"	40 seconds ago	Exited (130) 4 seconds ago	
	itso	ppc64le/debian:latest	"/bin/bash"	20 minutes ago	Exited (130) About a minute ago	
	e022e2a9cb66	ppc64le/debian:latest	"/bin/bash"	27 minutes ago	Exited (130) 20 minutes ago	
	angry_swartz	ppc64le/debian:latest	"/bin/bash"	28 minutes ago	Exited (0) 28 minutes ago	
	926963dbc80d	ppc64le/debian:latest	"/bin/bash"	29 minutes ago	Exited (0) 29 minutes ago	
	mad_lumiere	ppc64le/debian:latest	"/bin/bash"			
	7118a9b8a56d	ppc64le/debian:latest	"/bin/bash"			
	trusting_shockley	ppc64le/debian:latest	"/bin/bash"			
	1428429888b1	ppc64le/debian:latest	"/bin/bash"			
	loving_albattani					

**Note:** If the user does not specify a name to the container, it creates one automatically for you, as those described above, as “*loving\_albattani*”, “*trusting\_shockley*”, “*mad\_lumiere*”, and “*angry\_swartz*”.

## Creating a container

If you decided to create the container manually, other than using the **run** command, you can do that by using the **docker create** command, as shown in Example 7-21.

*Example 7-21 Creating a Docker container based on an image*

```
$ docker create --name itso2 ppc64le/debian
4960fbf70982c9e690e56b2c4789c8af76610f09b0add7a1a225a228140bb3c7
```

The advantage of using the *create* method over the *run* method is the ability to define detailed options for the container, as memory usage, CPU binding, and so on.

These are the most used options:

<b>--cpuset-cpus</b>	The host CPUs that are allowed to execute the container
<b>--hostname</b>	The container host name
<b>--ipc</b>	The IPC namespace
<b>--memory</b>	The amount of memory allocated to a specific container
<b>--expose</b>	The TCP/IP ports that are exposed
<b>--mac</b>	The mac address for the guest (Depends on the network option you are using)

**Note:** The arguments for any Docker command should be passed before the Docker image. Otherwise, it will be seen as a command. The argument position is not interchangeable.

## Starting the container and attaching to the console

Once you have the container created and want to start it, you can call the **start** command. To get the console terminal, you can also use the **attach** command, as shown in Example 7-22.

*Example 7-22 Getting the Docker console*

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS            PORTS              NAMES
$ docker start itso
itso

$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS            PORTS              NAMES
5c1da35da075       ppc64le/debian:latest  "/bin/bash"        2 days ago
Up 2 seconds                               itso

$ docker attach itso
root@5c1da35da075:/#
```

**Note:** If you try to attach to a container that is not started, Docker complains by using the following error message:

```
time="2015-11-16T09:29:10-05:00" level=fatal msg="You cannot attach to a stopped container, start it first"
```

## Changes to the original container image

After you have a Docker image running, all the image modification is saved in a different layer over the original image, and you can manage these changes by committing the changes, reverting, and so on.

Example 7-23 shows the commands that were executed in an image after it was created.

*Example 7-23 Docker image changes*

---

```
$ docker history ppc64le/debian
IMAGE          CREATED          CREATED BY
SIZE
ba92052e52c6   4 days ago      /bin/sh -c #(nop) CMD ["/bin/bash"]
0 B
8a78fb91edd3   4 days ago      /bin/sh -c #(nop) ADD
file:b20760af6fc16448d0 127.6 MB
```

---

To make an image change, you should do it through the container itself, instantiate the image in a container, change the container image, and commit this change to a new image. You can, later, replace the old image with the newer one.

Suppose you want to change the container image called `itso`, adding a new sub directory `foo` at `/tmp/` directory. To do it, you need to create a console, attach to the console, and run the `mkdir /tmp/foo` command.

When you do it, you can see the file system change by using the `docker diff` command. If you agree with the changes, you need to commit the changes to a new image by using the `docker commit` command, which is going to create a new image for you, based on the previous one, using the layer support from AUFS. This whole process is described in Example 7-24. After the commit image is generated, a new image is generated, with the commit ID `a98c7e146562`.

*Example 7-24 Committing a file system change to a new image*

---

```
$ docker diff itso
C /tmp
A /tmp/itso

$ docker commit itso
a98c7e1465623a6c8cf30b34b4038c14d3d04264fe818b89a4c3bcde33e84a36

$ docker images
REPOSITORY    TAG          IMAGE ID          CREATED          VIRTUAL SIZE
<none>        <none>       a98c7e146562     33 seconds ago  127.6 MB
ppc64le/debian latest       ba92052e52c6     4 days ago      127.6 MB
```

---

You can give a name to any image by using the `docker tag` command (Example 7-25).

*Example 7-25 Renaming a Docker image*

---

```
$ docker images
REPOSITORY    TAG          IMAGE ID          CREATED          VIRTUAL SIZE
<none>        <none>       a98c7e146562     14 minutes ago  127.6 MB

$ docker tag a98c7e146562 itso_version2

$ docker images
REPOSITORY    TAG          IMAGE ID          CREATED          VIRTUAL SIZE
itso_version2 latest       a98c7e146562     15 minutes ago  127.6 MB
```

---

## 7.6.4 Uploading your image to Docker hub

When you have a container image ready, you can upload it to Docker hub to be remotely available to others. Having an image at Docker hub allows you to download it from anywhere you want. It also helps you to deploy the same workload on any new container host.

### Docker login

In order to upload your image to Docker hub, you need to have a login at Docker hub service. You can create one easily and for no charge at <https://hub.docker.com>.

When you have a Docker hub login, you can associate your system to a Docker hub account by using the **docker login** command, as shown in Example 7-26.

*Example 7-26 Login information about Docker hub*

---

```
$ docker login
WARNING: The Auth config file is empty
Username: itso
Password:
Email: itso@itso.ibm.com
WARNING: login credentials saved in /home/itso/.dockercfg.
Login Succeeded
```

---

**Note:** After you enter your login information for the first time, it is saved at `~/.dockercfg` file. It saves your user ID and the encrypted password.

### Docker push

To be able to upload the image to Docker hub, you need to rename it appropriately. The container image name should have your Docker hub login as part of the image. For example, if your login is *username*, and the image name is *itso*, the image name should be *username/itso*. Otherwise, you get the following error:

```
time="2015-11-16T11:17:08-05:00" level=fatal msg="You cannot push a \"root\" repository. Please rename your repository to <user>/<repo> (ex: <user>/itso2)"
```

Example 7-27 shows a successful image upload to Docker hub. The image was originally called *itso*, but we need to rename it to *username/itso*, and pushed to the Docker hub.

*Example 7-27 Uploading an image to Docker hub*

---

```
$ docker push username/itso
The push refers to a repository [username/itso] (len: 1)
a98c7e146562: Image already exists
ba92052e52c6: Image successfully pushed
8a78fb91edd3: Pushing [==> ] 2.621 MB/50.58 MB
8a78fb91edd3: Image successfully pushed
Digest: sha256:476590f21a84f5fcc105beab5a1f29ec0970fd3474c2b2a5c0f8a6a0c74b3421
```

---

After the image is uploaded to the Docker hub service, you are able to see it by using the **docker search** command, as shown in Example 7-28.

*Example 7-28 Finding the image previously uploaded*

---

\$ docker search itso				
NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
username/itso		0		

---

## 7.6.5 Creating image from scratch

If you want to create your image from scratch, you can do it by using different methods depending on the type of image you need, and depending on the operating system.

The easiest way to do so is by creating a `.tar.gz` file with the root file system and importing it in Docker by using the **docker import** command. This is the method that is explained in this section. Two operating systems are used to describe how to create an image from scratch, Ubuntu and Debian.

### Ubuntu Core

Ubuntu has a distribution called *Ubuntu Core*, which is an image already stripped down for small environments like a Docker container. It is released with the Ubuntu releases, and contains around 50 MB compressed (172 MB decompressed) for each release. It also contains the most 100 important basic packages in Ubuntu. For more information about Ubuntu Core, see the following site:

<https://wiki.ubuntu.com/Core>

Because Ubuntu Core is available in a `.tgz` format in the web, we can point Docker to create an image from it by using a one-line command, as shown in Example 7-29.

*Example 7-29 Importing Ubuntu Core from the web*

---

\$ docker import				
http://cdimage.ubuntu.com/ubuntu-core/releases/15.10/release/ubuntu-core-15.10-core-ppc64el.tar.gz				
Downloading from http://cdimage.ubuntu.com/ubuntu-core/releases/15.10/release/ubuntu-core-15.10-core-ppc64el.tar.gz				
d13659b20bcebc5a89a8a90f4c811f09df0f013fb60896ed50eacaa8ec59d82c52.86 MB/52.86 MB				
Importing [=====>] 52.86 MB/52.86 MB				

---

When you have imported the Ubuntu Core, you now have an image for Ubuntu Core. You can rename it to “Ubuntu Core” and start it as Example 7-30 shows.

*Example 7-30 Renaming an image and starting it*

---

\$ docker images				
REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
<none>	<none>	24215bea40ea	11 minutes ago	172.1 MB

\$ docker tag 24215bea40ea ubuntucore				
---------------------------------------	--	--	--	--

\$ docker images				
REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
ubuntucore	latest	24215bea40ea	12 minutes ago	172.1 MB

```
$ docker run -t -i ubuntucore /bin/bash
root@6969df1e3eb3:/# cat /etc/os-release
NAME="Ubuntu"
VERSION="15.10 (Wily Werewolf)"
ID=ubuntu
```

```
ID_LIKE=debian
PRETTY_NAME="Ubuntu 15.10"
VERSION_ID="15.10"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
```

---

## Debootstrap

Debian and Ubuntu have a command to create a small operating system rootfs called **debootstrap**. It is an application that needs to be run on a Debian or Ubuntu system. It downloads a set of packages from an archive and decompresses the packages in a directory. The set of packages is defined in the command line, and in this example the minbase, which includes the minimal set of the most important packages. Ubuntu and Debian has the same package set, so, in order to differentiate Debian and Ubuntu, you basically need to point to the Debian or Ubuntu archive. It does not matter if you are in Ubuntu or Debian, you can create a cross operating system root file system.

To use it, you need to specify the platform you want (*ppc64el* for POWER8), as the distro and the local directory to decompress the files, as for example, the command below created a *localdirectory* directory, and install the minimal Debian (*minbase*) distribution using the *unstable* packages. Example 7-31 shows part of the expected output:

```
# debootstrap --arch=ppc64el --variant=minbase unstable localdirectory
http://ftp.debian.org/debian
```

*Example 7-31 Debootstrapping Debian in a local directory*

---

```
# debootstrap --arch=ppc64el --variant=minbase unstable localdirectory
http://ftp.debian.org/debian
[sudo] password for ubuntu:
W: Cannot check Release signature; keyring file not available
/usr/share/keyrings/debian-archive-keyring.gpg
I: Retrieving Release
I: Retrieving Packages
I: Validating Packages
I: Resolving dependencies of required packages...
I: Resolving dependencies of base packages...
I: Found additional required dependencies: adduser dmsetup insserv libapparmor1
libaudit-common libaudit1 libbz2-1.0 libcap2 libcap2-bin libcryptsetup4 libdb5.3
libdebconfclient0 libdevmapper1.02.1 libgcrypt20 libgpg-error0 libkmod2
libncursesw5 libsemanage-common libsemanage1 libsystemd0 libudev1 libustr-1.0-1
systemd systemd-sysv
I: Found additional base dependencies: binutils bzip2 cpp cpp-5
debian-archive-keyring dpkg-dev g++ g++-5 gcc gcc-5 gnupg gpgv libapt-pkg4.16
libasan2 libatomic1 libc-dev-bin libc6-dev libcc1-0 libdpkg-perl libgcc-5-dev
libgdbm3 libgmp10 libgomp1 libisl13 libitm1 libmpc3 libmpfr4 libreadline6
libstdc++-5-dev libstdc++6 libubsan0 libusb-0.1-4 linux-libc-dev make patch perl
perl-modules readline-common xz-utils
I: Checking component main on http://ftp.debian.org/debian...
I: Validating libacl1 2.2.52-2
I: Validating adduser 3.113+nmu3
I: Validating libapparmor1 2.10-2+b1
I: Validating apt 1.0.10.2
I: Validating libapt-pkg4.16 1.0.10.2
I: Validating libattr1 1:2.4.47-2
I: Validating libaudit-common 1:2.4.4-4
```



```
I: Validating libaudit1 1:2.4.4-4
I: Validating base-files 9.5
I: Validating base-passwd 3.5.38
I: Validating bash 4.3-14
I: Validating binutils 2.25.51.20151113-1
I: Validating build-essential 12.1
I: Validating bzip2 1.0.6-8
I: Validating libbz2-1.0 1.0.6-8
I: Retrieving libdebconfclient0 0.196
I: Validating libdebconfclient0 0.196
I: Validating coreutils 8.23-4
I: Validating libcryptsetup4 2:1.6.6-5
I: Validating dash 0.5.7-4+b2
I: Validating libdb5.3 5.3.28-11
I: Validating debconf 1.5.58
I: Validating debconf-i18n 1.5.58
I: Validating debian-archive-keyring 2014.3
I: Validating debianutils 4.5.1
I: Validating diffutils 1:3.3-2
I: Validating dpkg 1.18.3
I: Validating dpkg-dev 1.18.3
I: Validating libdpkg-perl 1.18.3
I: Validating e2fslibs 1.42.13-1
I: Validating e2fsprogs 1.42.13-1
I: Validating libcomerr2 1.42.13-1
I: Validating libss2 1.42.13-1
I: Validating findutils 4.4.2-10
I: Validating gcc-4.8-base 4.8.5-1
I: Validating gcc-4.9-base 4.9.3-5
I: Validating cpp-5 5.2.1-23
I: Validating g++-5 5.2.1-23
...
```

---

After the **debootstrap** command finishes to install the packages at the `localdirectory` directory, you are going to see a minimal Debian installed at this directory. After that, you can compress that directory on a file and use it to import in Docker, as shown in Example 7-32.

*Example 7-32 Importing from a local tgz file*

---

```
$ cd localdirectory
$ sudo tar -zcf ../debian-unstable.tgz *
$ cat ../debian-unstable.tgz | docker import -
ab90127bcd9308574948ef3920939ab9999bbc78f1ff3fe0
```

---

For more information about **debootstrap** and how to add extra packages to the image, check:

<https://wiki.debian.org/Debootstrap>





# PowerKVM Development Kit

This chapter covers the development kit and some development examples for the IBM PowerKVM product using C and Python programming languages.

- ▶ Introduction
- ▶ Development kit installation
- ▶ Connection and VM status
- ▶ Advanced operations

## 8.1 Introduction

The development kit is a software stack able to interact with PowerKVM virtualization capabilities, helping developers to have a fine-grained control of the hypervisor. All the development kit packages are open source and are easily found on any other common Linux distributions.

With the development kit installed, users can either extend the hypervisor features or create new ones.

You can write applications in the most common computer languages, as the main ones:

- ▶ C
- ▶ C++
- ▶ Perl
- ▶ PHP
- ▶ Python

This chapter covers the Libvirt API, which provides an extensive virtualization API with bindings to several programming languages. All examples in this chapter are written in C and Python.

### 8.1.1 Libvirt API

Libvirt provides all the API functions to do full PowerKVM management, which includes the following virtual machine operations:

- ▶ Create
- ▶ Control
- ▶ Modify
- ▶ Monitor
- ▶ Provision
- ▶ Start, resume, and stop

It is also possible to do hypervisor operations, which include:

- ▶ Network interface and bridge operations
- ▶ Firewall
- ▶ Disk and storage pool management
- ▶ General host management

Other than that, the libVirt API can be accessed locally and remotely using security protocols.

### Glossary

To understand the Libvirt API, it should make clear the names for each structure in a hypervisor machine.

#### **Node**

A node is the physical machine that runs the PowerKVM hypervisor.

#### **Hypervisor**

A hypervisor is the software stack that is able to virtualize a node, providing for example, processor, memory, and I/O support for the domains. All functions that you call for a certain domain are executed by a hypervisor driver. In PowerKVM the hypervisor driver is *qemu*.

## ***Domains***

A domain is the virtual machine created by the hypervisor on a specific node. It is also known as a *guest*.

## **8.2 Installation**

The development kit installation can be done in two different ways, either by installing from a public repository, or from the ISO image that is shipped with the POWER machines.

### **Installing from the repository**

The easiest way to install the PowerKVM development kit is by using the *yum* package manager. Internet access is required to connect to the official *yum* repository at the IBM site.

To validate that you have the proper repository working, you can check it with the **yum repolist -v** command. You should be able to see the IBM public repository listed, as shown in Example 8-1.

*Example 8-1 Assuring the online repository is configured*

---

```
# yum repolist -v
Loading "fastestmirror" plugin
Config time: 0.012
Yum version: 3.4.3
Loading mirror speeds from cached hostfile
Setting up Package Sacks
pkgsack time: 0.002

Repo-id      : powerkvm-updates
Repo-name    : IBM PowerKVM 3.1.0 - ppc64le
Repo-revision: 1445348973
Repo-updated : Tue Oct 20 13:49:33 2015
Repo-pkgs    : 0
Repo-size    : 0
Repo-baseurl :
http://public.dhe.ibm.com/software/server/POWER/Linux/powerkvm/release/3.1.0/updates/
Repo-expire  : 21,600 second(s) (last: Wed Nov  4 12:24:22 2015)
Repo-filename: /etc/yum.repos.d/base.repo

repolist: 1,641
```

---

Within that repository installed, there are two package groups, one named *Development-Tools-Basic* and another named *Development-Tools-Full*. Run the **yum grouplist** command, as shown in Example 8-2 to see those groups.

*Example 8-2 Development packages*

---

```
# yum grouplist
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
Installed groups:
    Base
    Virtualization
Available Groups:
    Development-Tools-Basic
    Development-Tools-Full
Done
```

---

### ***Basic development packages***

The basic development packages provide the basic development tools to start PowerKVM development. It contains around 23 packages and consumes around 133 MB of disk space. The list below shows the major packages:

- ▶ C compiler (GCC version 4.8.3)
- ▶ Version control system (Git version 1.8.3)
- ▶ RPM tools
- ▶ Build tools (automake version 1.13.4, libtool version 2.4.2, and autoconf version 2.69)
- ▶ Development libraries (python, perl, binutils, elfutils, audit, and others)

To install the basic development packages, run:

```
# yum groupinstall Development-Tools-Basic
```

### ***Full development packages***

If the basic development packages group does not contain all the packages needed, install a large set of development packages provided by the *Development-Tools-Full* group.

This group contains around 503 packages and consumes almost 300 MB of disk space when installed.

These are the major packages on these groups, among others:

- ▶ SystemTap
- ▶ Email stack
- ▶ Databases
- ▶ PHP interpreter
- ▶ Go interpreter
- ▶ Boost library

To install the full development packages, run:

```
# yum groupinstall Development-Tools-Full
```

For more information about Systemtap on POWER, check *SystemTap: Instrumenting the Linux Kernel for Analyzing Performance and Functional Problems*, REDP-4469.

## Installing package individually

You might want to avoid installing the full package set, and prefer to install just the packages you need. It can be done using yum also. For example, if you want to install just the gcc compiler, you might want to run a command like this:

```
# yum install gcc
```

## Installing from the ISO/DVD image

It is also possible to install the development kit from a DVD or ISO image. To do it, you need to mount the ISO or the DVD, create a repository file and use yum as explained above to download the packages.

If you have the PowerKVM DVD inserted in the disc tray, you might be able to find it in /mnt/cdrom. If you have just the ISO image, you can mount it at the same place using the following command:

```
# mount -o loop IBM-powerKVM-3.1.0.iso /mnt/cdrom
```

When you have the image at the /mnt/cdrom directory, you need to create a repository file to point to it. To do it, you need to create a repository file at /etc/yum.repos.d. You can name it for example local\_iso.repo and it should contain a content similar to Example 8-3.

*Example 8-3 Repository file for development kit installation from ISO*

---

```
[powerkvm-iso]
name=IBM PowerKVM $ibmver - $basearch - ISO media
baseurl=file:///mnt/cdrom/packages
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-ibm_powerkvm-$ibmver-$ibmmilestone
skip_if_unavailable=1
```

---

## 8.3 Architecture

Libvirt uses a set of pointers and structures that are worth mentioning. The main structure is called *virConnect* and it is generated when a connection to the hypervisor is created.

In order to create a connection with a node you want, use one of the three basic connection functions shown at Example 8-4.

*Example 8-4 Functions used to open a connection*

---

```
virConnectPtr virConnectOpen(const char *name)
virConnectPtr virConnectOpenReadOnly(const char *name)
virConnectPtr virConnectOpenAuth(const char *name, virConnectAuthPtr auth, int
flags)
```

---

These three functions are used initially to connect to the libvirt host in order to start the host and guest management. You can choose three different connection functions depending on what aspect of PowerKVM you want to manage.

This is a quick explanation about these three functions that you want to use to connect to a PowerKVM hypervisor:

- ▶ `virConnectOpen`

Used to connect to a hypervisor with full write access, if running on a local hypervisor or on a system without required authentication.

- ▶ `virConnectOpenReadOnly`

Used to connect to a hypervisor without authentication for read access. If you open a connection with this status, you have access to a restricted set of API.

- ▶ `virConnectOpenAuth`

Connect to a hypervisor using authentication. If you want just read-only access, you might use a `VIR_CONNECT_RO` flag during the function call.

If you want to connect your software to manage the *localhost* hypervisor, you might use an empty name argument. Other than that, you need to provide the URI for the remote machine that you want access to. Here is an example that you might want depending on how you configured PowerKVM.

For remote access, there are a couple different authentication modes:

- ▶ TLS/x509

This is going to require a *URI* as `qemu://example.ibm.com/system`

- ▶ SASL/Kerberos

For this method, you want to use URI as `qemu+tcp://example.ibm.com/system`. In order to use this option, you might want to enable libvirt to listen to TCP port 16509. This is an example on how to access libvirt using this method.

#### *Example 8-5 SASL access*

---

```
import libvirt
from getpass import getpass
mydata = "Hello"

def getCred(creds, data):
    print "yes"
    print str(creds)
    for cred in creds:
        print cred[1] + ": ",
        if cred[0] == libvirt.VIR_CRED_AUTHNAME:
            cred[4] = "myuser"
        elif cred[0] == libvirt.VIR_CRED_PASSPHRASE:
            cred[4] = "mypassword"
        else:
            return -1
    return 0

uri = "qemu+tcp://sid.ltc.br.ibm.com/system"
conn = libvirt.openAuth(uri,
                        [[libvirt.VIR_CRED_AUTHNAME, libvirt.VIR_CRED_PASSPHRASE],
                         getCred,
                         mydata], 0)
```

---



To enable it, you might change the configuration file `/etc/libvirt/libvirtd.conf`, mainly the `listen_tcp` and `listen_tls` options. The password should also be added by using the `saslpasswd2` command. An example can be something like:

```
# saslpasswd2 -a libvirt myuser
```

► SSH tunnelled

For this method, you use a URI as `qemu+ssh://user@example.ibm.com/system`. If you have keys on the remote Secure Shell (SSH) server, you are able to access the service without passphrase.

If you do not provide any URI, it uses the default one got from the `/etc/libvirt.conf` file in the `uri_default` file in the local machine (not on the PowerKVM hypervisor), as:

```
uri_default = "qemu:///system"
```

After connecting to the hypervisor, the function returns a pointer to a complex data structure that contains all the basic information needed to do the development that you want. This pointer is called `virConnectPtr` and points to a structure that is called `virConnect`. Figure 8-1 shows a little bit of the structure that you have when the connection is successful.

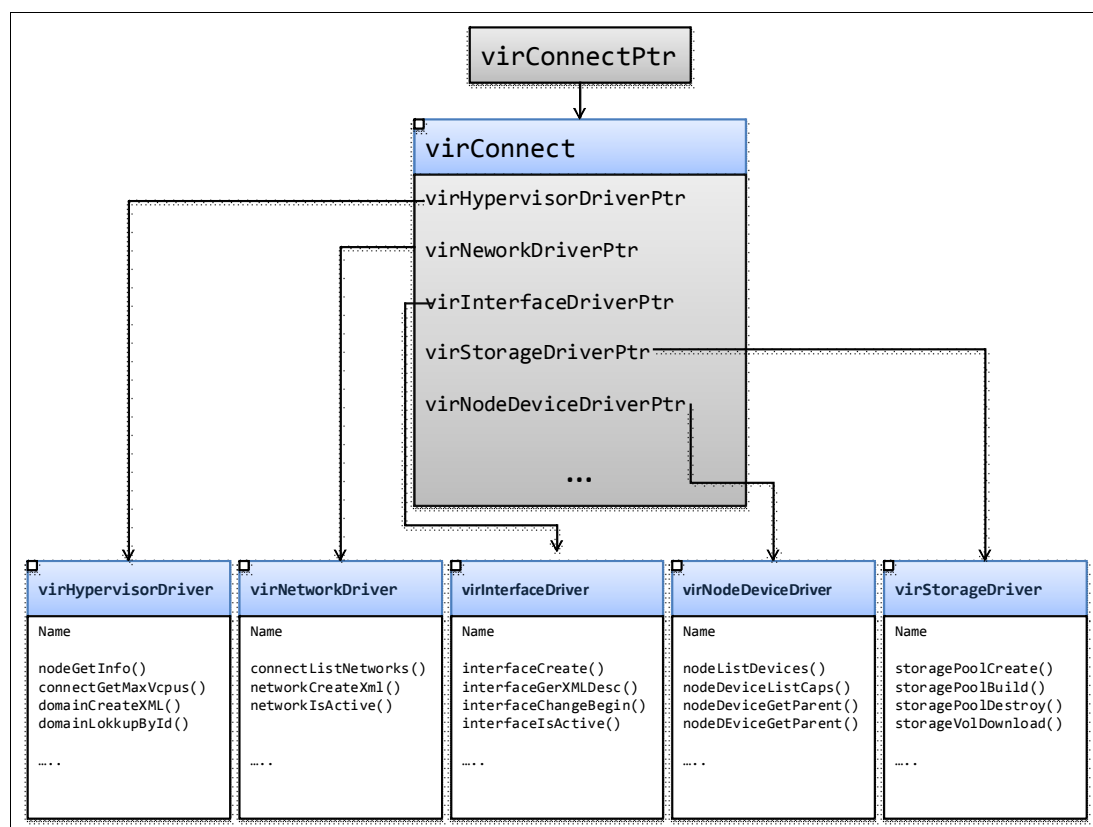


Figure 8-1 `virConnect` structure

## 8.4 Initial example

This first example covers a simple application implemented in C and Python. In this first example, you learn how to connect to a libvirt hypervisor using the `openReadOnly()` function and search for a virtual machine using the `lookupByName()` function.

## 8.4.1 Using Python binding

Example 8-6 shows a full application that is able to connect to an IBM PowerKVM hypervisor, and shows if a given virtual machine exists, and if it is running or powered down. The full details about this function are explained in 8.4.2, “Using the API in C” on page 235.

*Example 8-6 First example in Python*

---

```
import libvirt
import sys

def connect_to_hypervisor():
    # Opens a read only connection to the hypervisor
    connection = libvirt.openReadOnly(None)
    if connection is None:
        print 'Not able to connect to the localhost hypervisor'
        sys.exit(1)

    return connection

def find_a_domain(conn, domain):
    try:
        # Search for a virtual machine (aka domain) on the hypervisor
        # connection
        dom = conn.lookupByName(domain)
    except:
        # Virtual machine not found, returning 0
        return 0

    return dom.ID()

if __name__ == "__main__":
    if len(sys.argv) < 2:
        print "Usage\n%s <virtual machine name>" % sys.argv[0]
        sys.exit(1)

    vm_name = sys.argv[1]
    con = connect_to_hypervisor()
    ID = find_a_domain(con, vm_name)

    # close the open connection
    con.close()

    if ID > 0:
        print "Domain %s is running " % vm_name
    elif ID < 0:
        print "Domain %s is NOT running " % vm_name
    else:
        # If ID is 0, it means that the virtual machine does not exist
        # on the hypervisor
        print "Domain %s not found on this hypervisor " % vm_name
```

---

To run this application, you can call it directly as `python <script_name.py>`, as shown in Example 8-7. In this same example, the command is trying to discover if the machine *debian* is on, and it is. After that, it tries to check for a nonexistent virtual machine called *foo* and in this case, the script returns that the virtual machine does not exist.

*Example 8-7 Running the first example*

---

```
$ python example1.py
Usage
example1.py <virtual machine name>

$ python example1.py  debian
Domain debian is running

$ virsh list

```

Id	Name	State
2	ubuntu1504	running
3	debian	running
4	ubuntu1510	running
5	dockerhub	running
9	fedora	running

```

$ python example1.py  foo
Domain foo not found on this hypervisor

```

---

## 8.4.2 Using the API in C

Same example as Example 8-6 on page 234 but written in C, using the standard libvirt API.

The first function is able to connect to a localhost hypervisor (Example 8-8). It is useful when you want to create a local application. You can use any domain that you want, as explained in Example 8-4 on page 231.

*Example 8-8 Connecting to a localhost hypervisor*

---

```
virConnectPtr connect_to_hypervisor(){
    virConnectPtr conn;

    // Opens a read only connection to the hypervisor
    conn = virConnectOpenReadOnly("");
    if (conn == NULL) {
        fprintf(stderr, "Not able to connect to the hypervisor\n");
        exit(1);
    }

    return conn;
}
```

---

When you have the connection opened to the hypervisor, you need to find the domain ID. In order to do so, use the *virDomainLookupByName()* function, as shown in Example 8-9.

*Example 8-9 Return a domain ID for a domain name*

---

```
int find_a_domain(virConnectPtr conn, char *domain){
    virDomainPtr dom;

    // Search for a virtual machine (aka domain) on the hypervisor
    // connection
    dom = virDomainLookupByName(conn, domain);
    if (!dom) {
        return 0;
    } else {
        return virDomainGetID(dom);
    }
}
```

---

When you have the domain ID, you are able to identify if the virtual machine is active, inactive, or even exists. If the domain ID is larger than zero, the virtual machine is active. If the domain ID is a negative number, the domain is inactive. If the domain is zero, the virtual machine does not exist, as returned from the *find\_a\_domain()* function shown in Example 8-9.

In order to compile Example 8-10, you should use the *-lvirt* option, which means that it will link to the libvirt shared objects as:

```
$ gcc -Wall -o example1 example1.c -lvirt
```

After that, you are able to call **example1** directly and see an output similar to Example 8-7 on page 235.

*Example 8-10 First example main body*

---

```
#include <stdio.h>
#include <stdlib.h>
#include <libvirt/libvirt.h>

int main(int argc, char *argv[])
{
    virConnectPtr conn;
    int id;
    char *vm_name;

    if (argc < 2){
        printf("Usage\n%s <virtual machine name\n", argv[0]);
        exit(1);
    } else {
        vm_name = argv[1];
    }

    conn = connect_to_hypervisor();
    id = find_a_domain(conn, vm_name);
    virConnectClose(conn);

    if (id > 0){
        printf("Domain %s is running\n", vm_name);
    } else if (id < 0){
```

```

        printf("Domain %s is NOT running\n", vm_name);
    } else {
        // If ID is 0, it means that the virtual machine does not exist
        // on the hypervisor
        printf("Domain %s not found on this hypervisor\n", vm_name);
    }

    return 0;
}

```

---

**Note:** If you do not install the development packages, specifically the libvirt development packages, you can get an error similar to the following:

```

example1.c:3:29: fatal error: libvirt/libvirt.h: No such file or directory
#include <libvirt/libvirt.h>
        ^
compilation terminated.

```

## 8.5 Query memory and CPU utilization example

In this example, you learn new libvirt functions, such as how to list all active virtual machines using the function *virConnectListDomains()*. You also learn how to collect virtual machine memory utilization information through the function *virDomainMemoryStats()*.

This function uses the functions *connect\_to\_hypervisor()* as described in Example 8-8 on page 235 and *find\_a\_domain()*, as described in Example 8-9 on page 236.

By using these functions, you are going to be able to get a hypervisor connection. When the connection is returned, you are able to go through the active domains and collect the memory utilization information. This is done by using the function *get\_guest\_info()*, as shown in Example 8-11.

---

*Example 8-11 Get the memory information for the active guests*

---

```

virDomainInfo *get_guest_info(virConnectPtr conn,
                              int numDomains,
                              int *activeDomains)
{
    int i;

    virDomainInfo *domain_array;
    virDomainInfo info;
    virDomainPtr dom;
    virDomainMemoryStatStruct memstats[VIR_DOMAIN_MEMORY_STAT_NR];
    memset(memstats,
           0,
           sizeof(virDomainMemoryStatStruct) * VIR_DOMAIN_MEMORY_STAT_NR);

    // the array used to return information about domains
    domain_array = (virDomainInfo*)malloc(sizeof(virDomainInfo) * numDomains);
    if (domain_array == NULL) {
        fprintf(stderr, "Cannot allocate memory to domain_array.\n");
        virConnectClose(conn);
        exit(1);
    }
}

```

```

    }

    for (i = 0 ; i < numDomains ; i++) {
        dom = virDomainLookupByID(conn, activeDomains[i]);

        // Grab information about the domain memory statistics
        // VIR_DOMAIN_MEMORY_STAT_NR means that you want all the mem info
        virDomainMemoryStats(dom,
                               (virDomainMemoryStatPtr)&memstats,
                               VIR_DOMAIN_MEMORY_STAT_NR,
                               0);

        // Get generic domain information
        virDomainGetInfo(dom, &info);

        // free domain resources
        virDomainFree(dom);

        memcpy(&domain_array[i], &info, sizeof(virDomainInfo));

        // Replace the maxMem field with the information we want.
        // Keeping it in the domain info to move with the domainID
        // during qsort()
        domain_array[i].maxMem =
            memstats[VIR_DOMAIN_MEMORY_STAT_MINOR_FAULT].val / 1024;
    }

    return domain_array;
}

```

---

When you understand the functions above, the second example is easy to read. You can find it at Example 8-12. This new example prints the collected information in an ordered way.

---

*Example 8-12 Second development example*

---

```

#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <libvirt/libvirt.h>

#define SCREEN_COLS 76

// Function used by qsort() to compare time slice
int compare(const void *ptr1, const void *ptr2){
    const virDomainInfo *ptr1p = ptr1;
    const virDomainInfo *ptr2p = ptr2;

    if (ptr1p->cpuTime > ptr2p->cpuTime)
        return -1;
    else if (ptr1p->cpuTime < ptr2p->cpuTime)
        return 1;

    return 0;
}

```

```

// The application starts here
int main(int argc, char *argv[])
{
    virConnectPtr conn;
    int numDomains, i;
    int *activeDomains;
    long long sum = 0;
    virDomainPtr dom;
    virDomainInfo info;
    virDomainInfo *fst_measure, *snd_measure;

    conn = connect_to_hypervisor();

    // Get the Number of guests on that hypervisor
    numDomains = virConnectNumOfDomains(conn);

    // Get all active domains
    activeDomains = malloc(sizeof(int) * numDomains);
    virConnectListDomains(conn, activeDomains, numDomains);

    // Print header
    printf("%-15s | %4s | %7s | %9s | %28s\n",
           "Name", "vCPU", "Memory", "Mem fault",
           "Proportional CPU Utilization");
    for (i = 0; i < SCREEN_COLS; ++i)
        printf("-");
    printf("\n");

    // Capture total CPU utilization and memory faults in two instants
    fst_measure = get_guest_info(conn, numDomains, activeDomains);

    // Sleep for 1 second in order to grab the VM stats during this time
    sleep(1);
    snd_measure = get_guest_info(conn, numDomains, activeDomains);

    // Putting the CPU and Memory differences in the virDomainInfo for qsort()
    for (i = 0 ; i < numDomains ; i++) {
        fst_measure[i].maxMem = snd_measure[i].maxMem - fst_measure[i].maxMem;
        fst_measure[i].cpuTime = snd_measure[i].cpuTime - fst_measure[i].cpuTime;
        sum += fst_measure[i].cpuTime;
    }

    // second measurement is useless from now on
    free(snd_measure);

    // sort the machines by the most used one
    qsort(fst_measure, numDomains, sizeof(virDomainInfo), &compare);

    // List all the domains and print the info for each
    for (i = 0 ; i < numDomains ; i++) {
        dom = virDomainLookupByID(conn, activeDomains[i]);
        virDomainGetInfo(dom, &info);

        // Print domain information

```

```

printf("%-*.*s | %4d | ", 15, 15, virDomainGetName(dom), info.nrVirtCpu);

// Print memory information
printf("%7d | %9lu | ", (int)info.memory / 1024,
        fst_measure[i].maxMem / 1024);

// Print CPU utilization percentage
printf("%-5.2f %%\n", (double)100 * fst_measure[i].cpuTime / sum);

// free domain resources
virDomainFree(dom);
}

// Let's free the malloced structures
free(fst_measure);
free(activeDomains);

virConnectClose(conn);

return 0;
}

```

Example 8-13 shows the output of the example detailed above. In this output, you see a list of virtual machines, the number of memory faults each has, and also the CPU utilization for each of them. This list is sorted by CPU utilization.

*Example 8-13 Output of the second example*

```

$ ./example2
Name          | vCPU | Memory | Mem fault | Proportional CPU Utilization
-----
dockerhub     | 1    | 4096   | 810       | 81.11 %
ubuntu1510    | 8    | 8192   | 10        | 11.88 %
debian        | 4    | 4096   | 20        | 0.41 %
1404.2        | 1    | 4096   | 0         | 0.47 %
fedora        | 4    | 16384  | 0         | 0.00 %

```

## 8.6 Query guest network information

This section explores how to query information from a guest and use that to interact with the host. This is the last example because programming is not the target of this book, but it challenges you to go further on how to extend the PowerKVM system.

Example 8-14 shows how to connect to the Libvirt and how to get the guest XML definition. The **virConnectOpenReadOnly** function opens a restricted connection, which is enough to get the XML file.

Then, in the **get\_domain\_xml** function, the XML file of a given guest is returned by **virDomainGetXMLDesc** and stored in **domain\_xml**, which is a simple pointer to the char pointer allocated in and returned by **virDomainGetXMLDesc**.

*Example 8-14 Initial program implementation*

```

#include <stdio.h>

```



```

#include <string.h>
#include <stdlib.h>
#include <libvirt/libvirt.h>

#define MAC_ADDRESS_LEN 17
#define IP_ADDRESS_LEN 15

typedef struct mac_addresses
{
    char addr[MAC_ADDRESS_LEN + 1];
    struct mac_addresses *next;
} mac_t;

virConnectPtr connect_to_hypervisor()
{
    // opens a read only connection to libvirt
    virConnectPtr conn = virConnectOpenReadOnly("");
    if (conn == NULL) {
        fprintf(stderr, "Not able to connect to libvirt.\n");
        exit(1);
    }
    return conn;
}

void get_domain_xml(const virConnectPtr conn,
                   const char *domain,
                   char **domain_xml)
{
    if (conn == NULL || domain == NULL) {
        fprintf(stderr, "get_domain_xml parameters are null.\n");
        exit(1);
    }

    // search for a guest (aka domain) on the hypervisor
    virDomainPtr dom = virDomainLookupByName(conn, domain);
    if (dom == NULL) {
        fprintf(stderr, "Guest %s cannot be found\n", domain);
        exit(1);
    }

    *domain_xml = virDomainGetXMLDesc(dom, VIR_DOMAIN_XML_UPDATE_CPU);

    virDomainFree(dom);
}

```

---

In Example 8-15, the **get\_macs** function is implemented. It receives the **domain\_xml** got from **get\_domain\_xml** and extracts all network interface MAC addresses found. Those addresses are stored as elements of the struct **mac\_t**, which is a linked list. The function returns a pointer to the first element in the list.

#### *Example 8-15 get\_mac implementation*

---

```

mac_t *get_macs(const char *domain_xml)
{
    if (domain_xml == NULL) {
        fprintf(stderr, "domain_xml parameter cannot be null.\n");
    }
}

```

```

        exit(1);
    }

    mac_t *macs = NULL;
    char tmp_mac[MAC_ADDRESS_LEN + 1];

    // read the XML line by line looking for the MAC Address definition
    // and insert that address, if found, into a list of mac address
    // to be returned
    char *xml = strdup(domain_xml);
    char *token = strtok(xml, "\n");
    while (token) {

        if (sscanf(token, "%*[ ]<mac address='%17s'", tmp_mac)) {
            mac_t *mac = (mac_t*)malloc(sizeof(mac_t));
            strncpy(mac->addr, tmp_mac, MAC_ADDRESS_LEN);
            mac->addr[MAC_ADDRESS_LEN] = '\0';
            mac->next = macs;
            macs = mac;
        }
        token = strtok(NULL, "\n");
    }
    free(xml);

    return macs;
}

```

---

Example 8-16 has the implementation of the **print\_ip\_per\_mac** function. It is responsible to loop through the MAC addresses, stored in the list, searching for its respective IP address in the `/proc/net/arp` table. If such an IP address exists, the program prints that line and goes to the next element until the last element is reached.

---

*Example 8-16 print\_ip\_per\_mac implementation*

---

```

void print_ip_per_mac(mac_t *head)
{
    if (head == NULL)
        return;

    char line[1024];
    mac_t *tmp = head;

    // open the arp file for read, this is where the IP addresses
    // will be found
    FILE *arp_table = fopen("/proc/net/arp", "r");
    if (arp_table == NULL) {
        fprintf(stderr, "Not able to open /proc/net/arp\n");
        exit(1);
    }

    // for each MAC address in the list, try to find the respective
    // IP address in the arp file, so print the MAC address and its
    // IP address to output
    printf("%-17s\t%s\n", "MAC Address", "IP Address");
    while (tmp != NULL) {
        while (fgets(line, 1024, arp_table)) {

```

```

        if (!strstr(line, tmp->addr))
            continue;

        printf("%s\t%s\n", tmp->addr, strtok(line, " "));
        break;
    }
    fseek(arp_table, 0, SEEK_SET);
    tmp = tmp->next;
}
fclose(arp_table);
}

```

---

Example 8-17 is the main function, where the program is orchestrated and the resources used are freed, including the libvirt connection. Notice that the program expects the guest name as the parameter.

*Example 8-17 Implementing the main function*

---

```

int main(int argc, char *argv[])
{
    // checks if all parameters are set
    if (argc != 2){
        printf("Usage: %s <guest name>\n", argv[0]);
        exit(1);
    }

    char *domain_xml = NULL;
    mac_t *macs = NULL;
    const char *vm_name = argv[1];
    virConnectPtr conn = connect_to_hypervisor();

    // locates the guest and gets the XML which defines such guest
    get_domain_xml(conn, vm_name, &domain_xml);

    // gets a list with all mac address found in that particular guest
    macs = get_macs(domain_xml);

    // prints all IP address found for that guest
    print_ip_per_mac(macs);

    // clean all resources
    while (macs != NULL) {
        mac_t *tmp = macs->next;
        free(macs);
        macs = tmp;
    }
    free(domain_xml);
    virConnectClose(conn);

    return 0;
}

```

---

Example 8-18 on page 244 finally shows how to compile the program and the results you get.

*Example 8-18 Showing the final results*

---

```
$ gcc -Wall -g -o example3 example3.c -lvirt
```

```
$ ./example3 MyGuest
```

MAC Address	IP Address
52:54:00:9c:74:28	192.168.122.218
52:54:00:13:96:48	192.168.122.80

---

The Python version of the program is even simpler as shown in Example 8-19.

*Example 8-19 Python implementation*

---

```
#!/usr/bin/env python
import libvirt
import sys
import re

def connect_to_hypervisor():
    conn = libvirt.openReadOnly(None)
    if conn is None:
        print 'Not able to connect to libvirt'
        sys.exit(1)

    return conn

def get_domain_xml(conn, domain):
    if conn is None or domain is None:
        print 'get_domain_xml parameters are None'
        sys.exit(1)

    try:
        dom = conn.lookupByName(domain)

    except:
        print 'guest not found'
        sys.exit(1)

    return dom.XMLDesc()

def get_macs(domain_xml):
    if domain_xml is None:
        print 'domain_xml is None\n'
        sys.exit(1)

    mac_regex = '([a-zA-Z0-9]{2}):){5}[a-zA-Z0-9]{2}'
    return [m.group(0) for m in re.finditer(mac_regex, domain_xml)]

def print_ip_per_mac(macs):
    arp_table = None
    try:
        arp_table = open('/proc/net/arp', 'r')

    except IOError:
        print 'Not able to open /proc/net/arp\n'
        sys.exit(1)
```

```

print '%-17s\t%s' % ('MAC Address', 'IP Address')
for line in arp_table:

    row = line.split()
    if row[3] in macs:
        print '%s\t%s' % (row[3], row[0])

arp_table.close()

if __name__ == '__main__':
    if len(sys.argv) != 2:
        print 'Usage: %s <guest name>' % sys.argv[0]
        sys.exit(1)

    conn = connect_to_hypervisor()
    domain_xml = get_domain_xml(conn, sys.argv[1])
    macs = get_macs(domain_xml)
    print_ip_per_mac(macs)
    conn.close()

```

---

Example 8-20 shows the results that you get by using the program implemented in Python. As expected, the result is the same from the program written in C language.

---

*Example 8-20 Program results*

---

```

% ./example3.py MyGuest
MAC Address      IP Address
52:54:00:9c:74:28 192.168.122.218
52:54:00:13:96:48 192.168.122.80

```

---

For more information about application development using the PowerKVM Development Kit, refer to libvirt documentation:

<http://libvirt.org/devguide.html>



# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

## IBM Redbooks

The following IBM Redbooks publications provide additional information about the topics in this book. Some of these publications might be available in softcopy only:

- ▶ *IBM Power Systems S812L and S822L Technical Overview and Introduction*, REDP-5098
- ▶ *IBM PowerVC Version 1.2 Introduction and Configuration*, SG24-8199
- ▶ *Managing Security and Compliance in Cloud or Virtualized Data Centers Using IBM PowerSC*, SG24-8082

You can search for, view, download, or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, from the Redbooks website:

[ibm.com/redbooks](http://ibm.com/redbooks)

## Online resources

These websites are also relevant as further information sources:

- ▶ PowerKVM on developerWorks  
[https://www.ibm.com/developerworks/community/wikis/home?lang=en\\_us#!/wiki/W51a7ffc4dfd\\_4b40\\_9d82\\_446ebc23c550/page/PowerKVM](https://www.ibm.com/developerworks/community/wikis/home?lang=en_us#!/wiki/W51a7ffc4dfd_4b40_9d82_446ebc23c550/page/PowerKVM)
- ▶ KVM performance: SPEC virt sc2013 benchmark  
[http://www.spec.org/virt\\_sc2013](http://www.spec.org/virt_sc2013)
- ▶ IBM Fix Central  
<https://www.ibm.com/support/fixcentral>
- ▶ Debian  
<http://www.debian.org>  
<https://wiki.debian.org>
- ▶ Fedora  
<https://getfedora.org>
- ▶ Red Hat Enterprise Linux  
<http://www.redhat.com/products/enterprise-linux>
- ▶ SUSE Linux Enterprise Server  
<https://www.suse.com/products/server>
- ▶ openSUSE  
<https://www.opensuse.org>

- ▶ Ubuntu
  - <http://www.ubuntu.com>
  - <http://wiki.ubuntu.com>
- ▶ libvirt, the virtualization API
  - <http://libvirt.org>
  - <http://wiki.libvirt.org>
- ▶ Petitboot web page
  - <https://www.kernel.org/pub/linux/kernel/people/geoff/petitboot/petitboot.html>
- ▶ Kimchi and Ginger project page on Github
  - <https://github.com/kimchi-project>
- ▶ OpenPOWER on Github
  - <https://github.com/open-power>
- ▶ Slimline Open Firmware (SLOF) source code
  - <http://www.openfirmware.info/SLOF>
- ▶ NVIDIA CUDA Tools & Ecosystem
  - <https://developer.nvidia.com/cuda-tools-ecosystem>
- ▶ Slimline Open Firmware (SLOF)
  - <http://www.openfirmware.info/SLOF>
- ▶ NTP Pool project
  - <http://www.pool.ntp.org/en>
- ▶ Information on SLIRP
  - <http://bit.ly/1mvdVfT>
- ▶ OpenStack
  - <http://docs.openstack.org>
  - <http://wiki.openstack.org>
- ▶ Open vSwitch
  - <http://openvswitch.org>
- ▶ SELinux Project
  - <http://selinuxproject.org>
- ▶ IBM Cloud Manager with OpenStack Documentation
  - <http://ibm.co/1cc5r7o>
- ▶ Docker Hub
  - <https://hub.docker.com>
- ▶ IBM Power Linux on Twitter @ibmpowerlinux
  - <https://twitter.com/ibmpowerlinux>
- ▶ Implementing IBM PowerVC with PowerKVM servers Techbook
  - <https://ibm.box.com/PowerVC123-on-PowerKVM>
- ▶ IBM PowerKVM Development Toolkit
  - [https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/W51a7ffc\\_f4dfd\\_4b40\\_9d82\\_446ebc23c550/page/PowerKVM%20Development%20Kit](https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/W51a7ffc_f4dfd_4b40_9d82_446ebc23c550/page/PowerKVM%20Development%20Kit)



## Help from IBM

IBM PowerKVM 3.1 documentation

[http://www.ibm.com/support/knowledgecenter/SSZJY4\\_3.1.0/kvm\\_welcome/kvm\\_welcome\\_version31.htm](http://www.ibm.com/support/knowledgecenter/SSZJY4_3.1.0/kvm_welcome/kvm_welcome_version31.htm)

Linux information for IBM systems

<https://www.ibm.com/support/knowledgecenter/linuxonibm/liaab/ic-homepage.htm>

IBM Support and downloads

[ibm.com/support](http://ibm.com/support)

IBM Global Services

[ibm.com/services](http://ibm.com/services)



**Redbooks**

## **IBM PowerKVM: Configuration and Use**

SG24-8231-01

ISBN 073844152X



(0.5" spine)

0.475" <-> 0.873"

250 <-> 459 pages







SG24-8231-01

ISBN 073844152X

Printed in U.S.A.

Get connected

