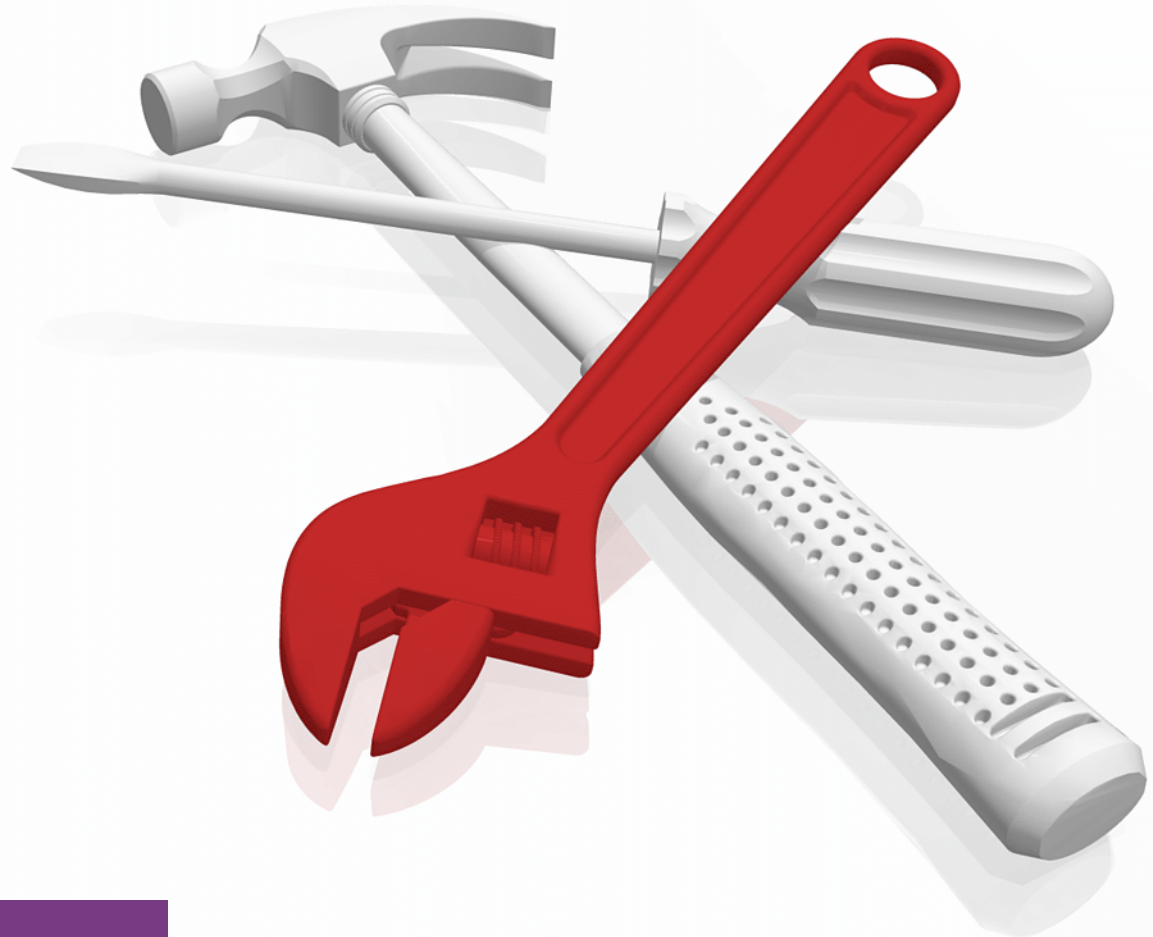


Virtualization Cookbook for IBM Z Volume 5 Kernel-based Virtual Machine

Octavian Lascu
Diogo Jose Basso Pigossi
Eduardo Simoes Franco
Eric Marins
Ewerson Palacio
Sergio Chang Mariselli



IBM Z

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

Second Edition (October 2022)

This edition applies to Red Hat REL 8.4, SUSE SLES 15 SP1, and Ubuntu 21.10 LTS.

© Copyright International Business Machines Corporation 2022. 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

Notices	ix
Trademarks	x
Preface	xi
Authors	xi
Now you can become a published author, too!	xii
Comments welcome	xii
Stay connected to IBM Redbooks	xiii
Chapter 1. Understanding the Kernel-based Virtual Machine on IBM Z	1
1.1 Kernel-based Virtual Machine on IBM Z	2
1.1.1 Why on IBM Z	2
1.1.2 KVM as a hypervisor on IBM Z	5
1.1.3 KVM on IBM Z running in a private cloud	5
1.2 KVM working on IBM Z	9
1.3 Managing and monitoring KVM on IBM Z	10
1.3.1 Libvirt	11
1.3.2 OpenStack	11
1.3.3 Virt-install	11
1.3.4 Virsh	12
1.3.5 Virt-manager	12
1.3.6 Cockpit	13
1.3.7 IBM Cloud Infrastructure Center	14
1.3.8 Platform management	15
1.3.9 Managing the KVM guest lifecycle	15
1.3.10 KVM host and guest monitoring	15
1.4 Securing KVM on IBM Z	16
1.4.1 Access control	16
1.4.2 IBM Secure Execution on IBM z15 and newer IBM Z systems	16
1.4.3 Authentication solutions	17
1.4.4 Multi-factor authentication	18
1.4.5 Audit	18
1.5 Availability with KVM on IBM Z	18
1.6 KVM on IBM Z backup and recovery	19
Chapter 2. Planning for the Kernel-based Virtual Machine host and guest	21
2.1 Basic requirements for KVM hosts and guests	22
2.1.1 Hardware requirements	22
2.1.2 Software requirements	23
2.1.3 Availability requirements	23
2.1.4 Deployment architecture	25
2.2 Planning resources for KVM guests	26
2.2.1 Compute considerations	27
2.2.2 Storage considerations	27
2.2.3 Network considerations	32
2.2.4 Encryption considerations	33
2.2.5 KVM guest domain considerations	34
2.2.6 Methods for installing Linux into a guest domain	35
2.2.7 Linux virtual machine live migration	36

2.3	Planning for management and monitoring	37
2.3.1	KVM host management	37
2.3.2	KVM host monitoring	38
2.3.3	KVM guest management	38
2.3.4	KVM guest monitoring	40
2.4	Planning for security	40
2.4.1	Access controls	40
2.4.2	Authentication solutions	41
2.4.3	Audit	41
2.4.4	Firewalls	42
2.4.5	Cryptography	42
2.4.6	Multi-factor authentication	43
2.5	Planning for backup and recovery	43
2.5.1	KVM host backups and recovery	44
2.5.2	KVM guest backup and recovery	45
Chapter 3. Preparing the Red Hat Enterprise Linux Kernel-based Virtual Machine environment for virtual machine use		47
3.1	Defining the target configuration	48
3.1.1	Logical View	48
3.1.2	Physical resources	49
3.1.3	Software resources	50
3.2	Preparing the infrastructure	50
3.2.1	Configuring the resources in Z platform	50
3.2.2	Configure the storage resources	50
3.3	Collecting information	53
3.3.1	Installing RHEL on an LPAR installation	54
3.3.2	Virtual machine installation information	55
3.4	Installing RHEL on an LPAR as KVM host	57
3.4.1	Preparing the installation	57
3.4.2	Installing RHEL on an LPAR	59
3.4.3	Preparing the host for virtualization	59
3.5	Configuring the KVM host	61
3.5.1	Defining NICs	61
3.5.2	Defining the bond interface	63
3.5.3	Defining HiperSocket interfaces	65
3.5.4	Defining HiperSocket interface to support VM guest network	66
3.5.5	Defining HiperSocket KVM host interface	67
3.5.6	Defining HiperSocket Converged interface	68
3.5.7	Defining SMC interfaces	69
3.5.8	Defining the MacVTap network	74
3.5.9	Defining crypto adapters and domains	75
3.6	Deploying virtual machines on KVM	80
3.6.1	Creating QCOW2 disk image file	80
3.6.2	Installing a new guest by using virt-install	80
3.6.3	Cloning a guest by using Virsh	82
3.6.4	Adding HiperSockets to the VM guest	83
3.6.5	Adding space to guest from ECKD DASD	84
3.6.6	Adding DASD space to guest as a VFIO device	86
3.6.7	Adding LUNs if FCP SCSI storage is used	89
3.6.8	Adding cryptography support to the VM guest	91
3.6.9	Using the Integrated Accelerator for zEnterprise Data Compression	92

Chapter 4. Preparing the SLES Kernel-based Virtual Machine environment for virtual machine use	95
4.1 Defining the target configuration	96
4.1.1 Logical View	96
4.1.2 Physical resources	97
4.1.3 Software resources	98
4.2 Preparing the infrastructure	98
4.3 Collecting information	102
4.3.1 Required information for SLES on an LPAR installation	103
4.3.2 Required information for VM installations	104
4.4 Installing SUSE on an LPAR as a KVM host	106
4.4.1 Preparing the installation	106
4.4.2 Installing SLES on an LPAR	107
4.5 Preparing the host for virtualization	108
4.6 Configuring the KVM host	111
4.6.1 Defining NICs	111
4.6.2 Defining the bond interface	112
4.6.3 Defining HiperSockets interfaces	114
4.6.4 Defining the HiperSocket interface to support VM guest network	115
4.6.5 Defining the HiperSocket interface of the KVM host	115
4.6.6 Defining HiperSocket Converged Interface	116
4.6.7 Defining SMC interfaces	117
4.6.8 Defining the MacVTap network	122
4.6.9 Defining the MacVTap network	122
4.6.10 Defining crypto adapters and domain	123
4.7 Deploying VMs on KVM	127
4.7.1 Creating QCOW2 disk image file	127
4.7.2 Installing a new guest by using virt-install	127
4.7.3 Cloning a guest by using Virsh	129
4.7.4 Adding HiperSockets to the VM guest	130
4.7.5 Adding space to guest from ECKD DASD	131
4.7.6 Adding DASD space to guest as a VFIO device	133
4.7.7 Adding LUNs when FCP SCSI storage is used	135
4.7.8 Adding cryptography support to the VM guest	136
4.7.9 Using the Integrated Accelerator for zEnterprise Data Compression	138
Chapter 5. Preparing the Ubuntu Kernel-based Virtual Machine environment for virtual machine use	141
5.1 Defining the target configuration	142
5.1.1 Logical View	142
5.1.2 Physical resources	143
5.1.3 Software resources	144
5.2 Preparing the infrastructure	144
5.2.1 Configuring resources	144
5.2.2 Configuring storage resources	144
5.2.3 Setting up the FTP server for the installation	146
5.3 Collecting information	146
5.3.1 Required information for Ubuntu on an LPAR installation	147
5.3.2 Required information for virtual machine installations	148
5.4 Installing Ubuntu on an LPAR as a KVM host	151
5.4.1 Preparing the installation	151
5.4.2 Installing Ubuntu on an LPAR	151
5.5 Preparing the host for virtualization	152

5.6	Configuring the KVM host	154
5.6.1	Defining NICs	154
5.6.2	Defining the bond interface	156
5.6.3	Defining HiperSockets interfaces	158
5.6.4	Defining HiperSocket interface to support VM guest network	158
5.6.5	Define HiperSocket Converged Interface	161
5.6.6	Defining SMC interfaces	162
5.6.7	Defining the MacVTap network	167
5.6.8	Defining crypto adapters and domain	168
5.7	Deploying virtual machines on KVM	173
5.7.1	Creating QCOW2 disk image file	173
5.7.2	Installing a new guest by using virt-install	173
5.7.3	Cloning a guest using Virsh	175
5.7.4	Adding HiperSockets to the VM guest	176
5.7.5	Adding space to guest from ECKD DASD	177
5.7.6	Adding DASD space to a guest as a VFIO device	179
5.7.7	Adding LUNs if you have FCP Storage	182
5.7.8	Adding cryptography support to the VM guest	184
5.7.9	Using the Integrated Accelerator for zEnterprise Data Compression	185
Chapter 6. Managing the Kernel-based Virtual Machine environment		187
6.1	Managing resources	188
6.1.1	Virsh	188
6.1.2	Virtual Machine Manager	190
6.1.3	Cockpit	192
6.1.4	OpenStack	194
6.1.5	Choosing the correct tool	196
6.2	Recovery management	197
6.2.1	Snapshot	198
6.2.2	Compressing data and backup	198
6.2.3	IBM FlashCopy	199
6.3	Security management	201
6.3.1	FreeIPA	201
6.3.2	sVirt	205
6.3.3	AppArmor	206
6.3.4	Linux Audit	207
Chapter 7. High Availability for IBM General Parallel File System		211
7.1	Environment overview	213
7.2	Zoning and LUN masking	214
7.3	Downloading IBM Spectrum Scale from IBM Fix Central	219
7.4	Installing IBM Spectrum Scale	221
7.4.1	Working with clusters and deploying protocols	223
7.4.2	Configuring IBM Spectrum Scale	224
7.5	Building the GPFS portability layer	225
7.6	Handling Linux kernel updates	226
7.7	GPFS general configuration	228
7.7.1	Installing the licensing	229
7.7.2	Validating or listing the cluster configuration	229
7.7.3	Displaying the state of GPFS cluster	230
7.7.4	Changing the range ports that are used for command execution	230
7.7.5	Configuring FCP Channels to all KVM Compute (GPFS cluster) servers	230
7.7.6	Using tiebreaker disks	240

7.7.7	Displaying the NSD information	241
7.8	Working with the General Parallel File System	241
7.8.1	Creating and configuring GPFS	241
7.8.2	Mounting and validating the GPFS	242
7.8.3	Configuring the SELinux file's context	242
7.8.4	Starting GPFS automatically	243
7.8.5	Tiebreaker disk recommendations	243
7.8.6	Setting up a tiebreaker disk	249
7.8.7	Enabling Persistent Reserve	249
Chapter 8.	Using IBM Secure Execution	253
8.1	Introduction to IBM Secure Execution	254
8.2	How IBM Secure Execution works	255
8.3	Enabling and verifying that the CPC is Secure Execution ready	255
8.3.1	Importing a key bundle into LinuxONE	255
8.4	KVM host and guest software requirements	256
8.5	Enabling an Ubuntu 20.04 LTS KVM host for IBM Secure Execution	257
8.6	Enabling an SLES 15 SP2 KVM host for IBM Secure Execution	258
8.7	Enabling an Ubuntu 20.04 KVM Guest for IBM Secure Execution	261
8.7.1	Installing a standard Linux guest on encrypted disk storage	261
8.7.2	Updating KVM guest /etc/crypttab to avoid entering a password at start	262
8.7.3	Editing the domain.xml to include iommu='on'	268
8.7.4	Obtaining the host key documents for the CEC	269
8.7.5	Validating the key material	269
8.7.6	Building a secured initrd image file by using genprotimg on KVM guest	271
8.7.7	Updating guest zipl to boot with secured image in IBM Secure Execution mode	272
8.7.8	Removing any boot option that is not in Secure Execution mode	276
8.7.9	Further Ubuntu guest hardening	276
8.7.10	Removing older unencrypted artifacts from the /boot partition	278
8.8	Enabling a SLES 15 SP2 KVM Guest for IBM Secure Execution	279
8.8.1	Installing a standard Linux guest on encrypted disk storage	280
8.8.2	Updating KVM guest /etc/crypttab to avoid entering a password at start	286
8.8.3	Editing the domain.xml to include iommu='on'	292
8.8.4	Obtaining the host key documents for the CEC	293
8.8.5	Validating the key material	293
8.8.6	Building a secured initrd image file by using genprotimg on KVM guest	293
8.8.7	Updating guest zipl to boot with secured image in IBM Secure Execution mode	294
8.8.8	Updating guest grub2 to boot with secured image in IBM Secure Execution mode	297
8.8.9	Removing any start option that is not Secure Execution mode	300
8.8.10	Further SLES 15 guest hardening	301
8.8.11	Removing unencrypted older artifacts from /boot/zipl and encrypted artifacts from /boot	302
8.9	Enabling a RHEL KVM Guest for Secure Execution	304
8.9.1	Installing a standard Linux guest on encrypted disk storage	305
8.9.2	Updating KVM guest /etc/crypttab to avoid entering password at boot	308
8.9.3	Editing the domain.xml to include iommu='on'	312
8.9.4	Obtaining the host key documents from the CEC	313
8.9.5	Validating the key material	314
8.9.6	Building a secured initrd image file using gemprotimg on KVM guest	316
8.9.7	Updating guest zipl to boot with secured image in Secure Execution mode	317
8.9.8	Securely removing the original unprotected kernel, initrd , and parmfile files	319
8.9.9	Further RHEL guest hardening	319

8.9.10 Removing unencrypted, older artifacts from /boot	320
Chapter 9. IBM Cloud Infrastructure Center on Kernel-based Virtual Machines . . .	321
9.1 Installing IBM Cloud Infrastructure Center	322
9.1.1 Before you install IBM Cloud Infrastructure Center.	330
9.2 Configuring IBM Cloud Infrastructure Center	335
9.2.1 Other tasks	336
9.2.2 Creating and adding images	344
9.2.3 Adding storage providers	349
9.2.4 Extending the root file system	352
9.2.5 User tasks	353
9.3 Creating a bond for KVM administration network interfaces	365
Appendix A. Live Virtual Server Migration	369
Introduction	370
Live migration phases	370
Provisioning two servers	371
Performing a live migration	375
Procedure	375
Restriction	376
Appendix B. Kernel-based Virtual Machine LPAR live migration.	379
Overview	380
Appendix C. Scripts for SLES guest installation	385
Preparing and setting up for AutoYAST installation	386
AutoYAST configuration file for KVM guest	387

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](http://www.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.

Db2®	IBM Z®	Tivoli®
DS8000®	IBM z15™	WebSphere®
FICON®	Parallel Sysplex®	z/Architecture®
FlashCopy®	QRadar®	z/OS®
GDPS®	Redbooks®	z/VM®
IBM®	Redbooks (logo)  ®	z15™
IBM Spectrum®	Storwize®	

The following terms are trademarks of other companies:

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

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

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Ansible, Red Hat, are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

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

Preface

This IBM® Redbooks® publication provides a broad explanation of the kernel-based virtual machine (KVM) on IBM Z® and how it can use the z/Architecture®. It focuses on the planning of the environment and provides installation and configuration definitions that are necessary to build, manage, and monitor a KVM on Z environment.

This publication applies to the supported Linux on Z distributions (Red Hat, SUSE, and Ubuntu).

This IBM Redbooks publication is useful to IT architects, system administrators, and users who plan for and install KVM on IBM Z. The reader is expected to have an understanding of IBM Z hardware, KVM, Linux on Z, and virtualization concepts.

Authors

This book was produced by a team of specialists from around the world working at IBM Redbooks, Poughkeepsie Center:

Octavian Lascu is an IBM Redbooks Project Leader and Senior IT Infrastructure Specialist at IBM Redbooks, Poughkeepsie Center.

Diogo Jose Basso Pigossi is a Mainframe Infrastructure Architect and Technical IT Specialist working with IBM since 2008. He holds a degree in Computer Science and an MBA in Information Security Management. His areas of expertise are Mainframe Configuration Management and zVM Base. For the last few years, Diogo and his team led the modernization of CIO data centers and the efforts to offer cloud solutions to Z hypervisors as IBM Cloud Infrastructure Center and KVM. Today, he works for Kyndryl CIO.

Eduardo Simoes Franco is an IT Specialist and Technology Consultant at IBM. He has more than 20 years of experience with IT Solutions, projects, and infrastructure support. He has held technical and management positions at several large corporations in servers support as a network analyst, security officer, IT coordinator, and consultant. Currently, he supports large IBM clients worldwide on Docker, virtualization, and Linux on IBM Z platform.

Eric Marins is a Senior IT Architect in Brazil, focused on hybrid cloud solutions, Infrastructure and Platform solutions, and competencies, including High Availability, Disaster Recovery, Networking, Linux, and Cloud. He has many years of experience working with and writing about IBM Z, Linux, and Open Source topics. He has co-authored more than seven IBM Redbooks publications.

Ewerson Palacio is an IBM Redbooks Project Leader. He holds Bachelors degree in Math and Computer Science. Ewerson worked for IBM Brazil for over 40 years and retired in 2017 as an IBM Distinguished Engineer. Ewerson co-authored many z Systems IBM Redbooks publications, and created and presented ITSO seminars around the globe.

Sergio Chang Mariselli is an IT Specialist at Kyndryl. He has more than 10 years of experience with IT infrastructure projects. He worked in IBM for 9 years and now is working in Kyndryl, leading IBM Z-related projects. In last 6 years, Sergio has been working with IBM z/VM® and Linux on Z. Currently, he leads projects, manages platforms, advises clients, and supports Peruvian clients on IBM Z.

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

IBM Redbooks, Poughkeepsie Center

Lydia Parziale
Robert Haimowitz

IBM

Richard Young
Viktor Mihajlovski
Boris Fiuczynski
Matthew Rosato
Bill Lamastro
Tom Ambrosio
Melissa Carlson
Dorothea Matthaues
Stefan Raspl

SUSE

Mike Friesenegger

Special thanks to the authors of the first edition:

Bill White, Sergio Chang Mariselli, David Borges de Sousa, Eduardo Simoes Franco, Pablo Paniagua, Richard Ruppel and Richard Young.

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 IBM Redbooks 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

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
- ▶ Send your comments in an email to:
redbooks@us.ibm.com
- ▶ Mail your comments to:

IBM Corporation, IBM Redbooks
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- ▶ 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>



Understanding the Kernel-based Virtual Machine on IBM Z

This chapter introduces the Kernel-based Virtual Machine (KVM) concepts and the key capabilities of IBM Z that KVM can use.

This chapter includes the following topics:

- ▶ 1.1, “Kernel-based Virtual Machine on IBM Z” on page 2
- ▶ 1.2, “KVM working on IBM Z” on page 9
- ▶ 1.3, “Managing and monitoring KVM on IBM Z” on page 10
- ▶ 1.4, “Securing KVM on IBM Z” on page 16
- ▶ 1.5, “Availability with KVM on IBM Z” on page 18
- ▶ 1.6, “KVM on IBM Z backup and recovery” on page 19

Terminology: The terms *guest*, *virtual server*, and *virtual machine* (VM) are interchangeable. These terms are used throughout this book, depending on the component that is described.

1.1 Kernel-based Virtual Machine on IBM Z

Virtualization allows businesses to address scale and performance demands while providing better use of compute resources. Businesses also came to rely on open source options to give standardized, cost-effective virtualization solutions.

The KVM is the open source virtualization option that is built into Linux distributions, such as Red Hat Enterprise Linux Server, SUSE Linux Enterprise Server, and Ubuntu, and supported on IBM Z. KVM running on a Linux operating system image¹ acts as a hypervisor, offering the ability to run many VMs or guests under a single host machine.

KVM includes the various operating system components that are needed to run guests, such as a memory manager, a process scheduler, I/O capabilities, device drivers, a network stack, and a security manager.

As a hypervisor, KVM can share and manage memory, CPUs, and I/O (storage and networks) between the VMs that are running on the host operating system (hypervisor). Over-committing of shared memory and CPU resources is possible to enable greater scalability.

Also, KVM includes live guest migration (LGM), which is the ability to move a running VM between hosts with minimal effect. The ability to dynamically add and remove CPU, memory, and virtual I/O devices also exist. These functions fit well with the overall high availability and resiliency capabilities of IBM Z.

From a Linux administrator perspective, KVM provides a standard set of Linux tools and interfaces, which feature a common user experience across various hardware platforms. This ability makes managing the IBM Z similar to any other compute resource (hardware platform).

Also, the use of standard open source interfaces provides the key to the integration of Linux on IBM Z, which helps optimize modern applications and accommodate scale-out clusters and scalable clouds. This use of standard open source interfaces includes access to a wide range of software packages that provide the suitable tools for building, testing, and deploying applications and services.

In addition to the common user experience, KVM can use the exceptional capabilities and qualities of service of the IBM Z platform in the areas of security and data protection.

1.1.1 Why on IBM Z

Linux on Mainframe environment dramatically reduces the IBM Z specific and expensive skills and permits Linux environments to gain all benefits of this platform, such as vertical scalability, resiliency, security, and interoperability.

For more than two decades, Linux workloads were supported on IBM Z. Over those years, IBM Z platforms continuously improved and enhanced performance, security, resiliency, and virtualization at all levels, from the hardware, through the firmware, to the software stack.

IBM z13 and higher offer large cache sizes, along with the Simultaneous Multi-Threading (SMT²) and Single Instruction Multiple Data (SIMD) instruction set. These features enable more efficient processing of large volumes of data while providing high-performance transaction processing and more analytics capabilities.

¹ The Linux operating system is installed in a logical partition (LPAR) that is running on an IBM Z / IBM LinuxONE Central Processing Complex (CPC).

² SMT available on z13 and newer IBM Z generations.

Also, superior I/O throughput is achieved with high-performance Fibre Channel connections. High-speed network capabilities provide exceptional performance with various options. External network connectivity is possible by using OSA-Express, and RoCE Express features as regular TCP/IP devices. Low latency communication also is possible with memory-to-memory options, such as HiperSockets or Shared Memory Communication (SMC) through Remote Direct Memory Access over Converged Ethernet (RoCE).

Security was always built into the IBM z/Architecture. In addition to high-performance, specialized encryption facilities on each processor chip by using Central Processor Assist Crypto Function (CPACF), separate tamper-resistant Crypto Express features can also be used for greater protection of encryption keys and sparing CPU cycles.

Virtualization and IBM Z capabilities help you meet the demanding security, availability, and scalability requirements of today's workloads. Along with the large number of logical partitions and VMs, IBM Z goes one step further, optimizing throughput with the use of Simultaneous Multithreading (SMT). This optimization allows more than one thread to run at the same time on the same core. SMT is supported by the Integrate Facility for Linux (IFL) processors on IBM Z that Linux and KVM use.

The IBM Z (IBM z15) included a new feature that the KVM hypervisor can use to bring an even higher level of protection and isolation to VM guests. This feature is known as IBM Secure Execution for Linux®. It provides the ability to protect a KVM guest's memory, boot image, and state information from the KVM hypervisor and other guests that are sharing this KVM host. This capability enables you to populate large KVM hosts with different guests from different departments or clients, which allows a more robust multi-tenant environment that helps reduce costs and increase resource use.

To improve performance and reduce CPU consumption, IBM z14 introduced the IBM Integrated Accelerator for zEnterprise® Data Compression (zEDC). This hardware feature accelerates the data compression process in the Linux machine and consequently reduces the cost of storing, processing, and transporting data.

IBM Z is known for its reliability, availability, and serviceability (RAS) capabilities. RAS is built into the architecture's hardware and software stacks, where the mean time between failures is measured in decades, making the availability of 99.999% possible.

The Linux operating system that is running in an IBM Z platform can consolidate hundreds of scaled-out distributed servers onto one machine. Because of IBM Z hardware architecture, it is possible to run thousands of VMs within a single system, which enables achieving some peaks of 100% system utilization.

With its extended hardware capabilities, IBM Z can handle 1 million Docker containers without slowing down. When Node.js and MongoDB are used, it can handle over 30 billion web events.

The IBM Z hardware is prepared to scale up and scale out, which fits the needs of the business and optimizes the infrastructure costs. It has a compelling total cost of ownership (TCO) and total cost of acquisition (TCA) metrics that show considerable cost savings over the x86 at comparable workloads. For more information, see this [web page](#).

Linux running on IBM Z reduces costs in the following ways:

▶ Reduced licensing fees:

Databases, operating systems, application servers, and management software in a current distributed server farm can be licensed cost-effectively by using the powerful IBM Z Integrated Facility for Linux (IFL). Because licensing fees are charged by core (similar to the distributed environment), the cost savings arise because IBM Z hardware can use processor time more efficiently than x86, and do the same amount of work on fewer cores.

▶ Reduced risk of downtime

The IBM Z and LinuxONE hardware merges the reliability, availability, and serviceability (RAS) characteristics from the previous performance-optimized servers and provides advanced features, such as the following examples:

- IBM Z servers operate with two sets of redundant power supplies. Each set of the power supplies has its individual power cords or pair of power cords.
- Processor and memory PU refresh, RAIM memory, and cache symbol ECC provide a robust computing platform.
- PU sparing, array macro sparing, and micro-array masking integrated sparing allow error detection, isolation, and reassignment of the faulty component without affecting the system.
- Dynamic oscillator switch-hover capability allows a backup oscillator to transparently detect a failure, switch-over and then, continue to provide the clock signal to the system.
- The wide use of redundancy in the power, cooling, and service network. A “power redundancy test” is provided so you can verify that the server is power redundant.

▶ Save energy and be green³

It can consolidate hundreds or thousands of servers in a single box. An IBM z15 T01 single system performance enables reduced overall system power consumption by 50% versus the equivalent x86 configuration. An IBM z15 single frame system requires 75% less floor space than compared x86 that is running the same workloads and throughput.

▶ Save labor costs

The complexity of maintenance of the environment is decreased because many servers can be virtualized in a single box.

▶ Optimize the storage and save storage costs

The IBM Integrated Accelerator for zEnterprise® Data Compression (zEDC) that is available on Linux on IBM z15™ can enable high-speed compression, which saves the amount of storage use, and reduces the CPU consumption and elapsed time.

Today, VMs are the core of cloud infrastructure as a service (Yeas). The new version of IBM Z continuously brings improvements, including several new features, which support the hybrid cloud infrastructure in a single box through high scalability, resilience, and security to host several types of workloads while maintaining the high availability and high performance of the entire IT environment.

For more information about all IBM Z platform capabilities, see this [IBM Z web page](#).

³ Refer to this [IBM web page](#).

1.1.2 KVM as a hypervisor on IBM Z

Hypervisors can be implemented in hardware or software and the IBM Z platform allows for both. All IBM Z platforms are delivered with the hardware hypervisor, which is known as Processor Resource/Systems Manager (PR/SM). PR/SM is implemented in the firmware and can virtualize and share system resources without extra software.

PR/SM also enables and requires defining and managing subsets of resources into logical partitions (LPARs). LPAR definitions include the number of logical processor units, the amount of memory, and the shared or dedicated I/O resources for storage and networks. The LPAR definitions can be changed dynamically to add or remove resources through the Hardware Management Console (HMC). A high-level overview of KVM on IBM Z is shown in Figure 1-1.

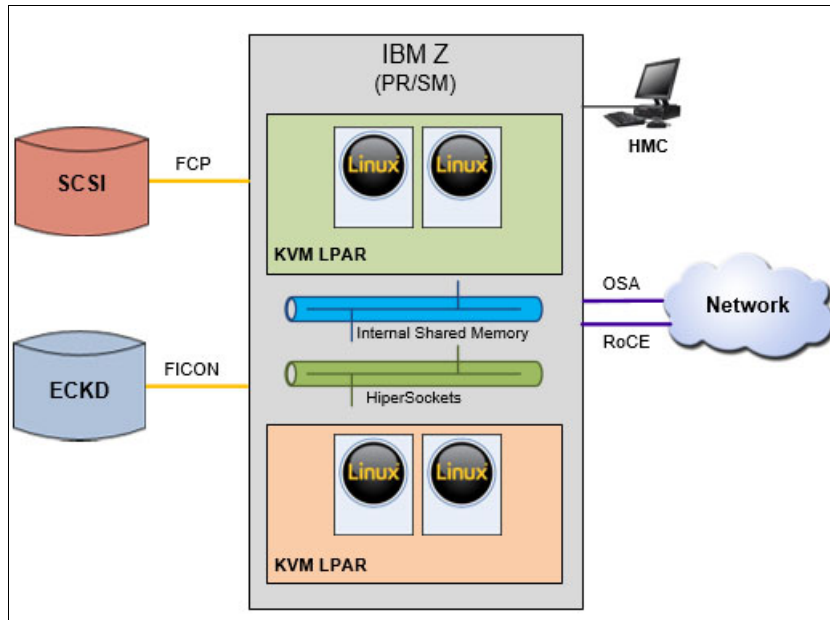


Figure 1-1 KVM running in IBM Z LPARs

1.1.3 KVM on IBM Z running in a private cloud

One of the strengths of KVM is the flexibility to adapt to multiple platforms. It has native hardware virtualization, allows open patterns, and meets all cloud requirements.

Combined with the huge hardware capacity that is available in the IBM Z, companies worldwide might not use only on-premises enterprise infrastructure. They still evolve to a modern private cloud model that is compliant with most cloud requirements with more robust security and reliability.

The Linux/KVM ecosystem is the entry portal of many modern applications in the cloud. Since the first version of IBM Z, the hardware resources virtualization (which is the core of cloud computing) was present natively on this platform.

KVM running in an IBM Z LPAR integrates seamlessly with PR/SM. KVM views the virtualized CPUs, memory, and I/O devices that are managed by PR/SM as real resources. VMs that request processing time is first handled by KVM and then passed to PR/SM for dispatching of the work to the physical CPU.

Storage connectivity

Two storage types (Small Computer System Interface [SCSI] and IBM Extended Count Key Data [ECKD]), are supported by KVM and Linux on IBM Z. Both types of storage are connected through IBM Fibre Connection (FICON®) features, which follow Fibre Channel (FC) technology standards. Internal NVMe storage also is supported on the IBM LinuxONE.

The FICON features support the following protocols:

- ▶ FICON
An enhanced protocol over FC that supports ECKD devices, including disks, tapes, and printers.
- ▶ Fibre Channel Protocol (FCP)
A standard protocol that supports SCSI devices (disk and tape).

Linux and KVM on IBM Z also supports other storage-related protocols, such as the following examples:

- ▶ Internet Small Computer Systems Interface (iSCSI)
This protocol allows client initiators to send SCSI commands to SCSI storage device targets on remote servers over TCP/IP.
- ▶ Spectrum Scale - GPFS
GPFS is a cluster file system. It provides concurrent access to a single file system or set of file systems from multiple nodes. These nodes can all be SAN attached or a mix of SAN and network attached, which enables high-performance access to this common set of data to support a scale-out solution or provide a high availability platform.
- ▶ Network File System (NFS) client
An NFS allows remote hosts to mount file systems over a network and interact with those file systems as though they are mounted locally.

Network connectivity

Network connectivity covers the interfaces between the IBM Z platform and external networks with Open Systems Adapter-Express (OSA-Express) and RoCE Express features and specialized internal interfaces for intra-system communication through IBM HiperSockets and Internal Shared Memory (ISM).

OSA-Express features

OSA-Express features provide industry-standard Ethernet local area network (LAN) connectivity and communication in a networking infrastructure. OSA-Express features use the IBM Z I/O architecture, called *queued direct input/output* (QDIO). QDIO is a highly efficient data transfer mechanism that uses system memory queues and a signaling protocol to exchange data directly between the OSA-Express microprocessor in the feature and the network stack that is running in the operating system.

To improve the network availability and bandwidth on IBM Z that uses a pair of physical OSA-Express ports, you can use the Link Aggregation Control Protocol (LACP), bonding mode 4. This LACP combines multiple network interfaces into a single logical interface that is called a *bond interface*.

The network interfaces that are aggregated together are called *slave devices*, and the bonded logical interface is called the *master device*. By using channel bonding, the multiple slave devices act as though only one master device works with more bandwidth and network redundancy.

Adding cards can result in increased throughput, particularly when the OSA card is being fully used. Measurement results show an increase in throughput from 6% to 15% for a low-utilization OSA card to an increase in throughput from 84% to 100% for a high-utilization OSA card and reductions in CPU time ranging from 0% to 22%.

Other forms of bonding exist, such as Transmit Load Balancing (TLB), bonding mode 5, where the current slave receives incoming traffic. If the receiving slave fails, another slave takes over the MAC address of the failed slave, and Adaptive Load Balancing (ALB), bonding mode 6, where, unlike LACP, ALB does not require any specific switch configuration.

The receiving packets are load balanced through ARP negotiation. TLB and ALB can achieve similar bandwidth without the LACP restrictions. For more information, see this [LIBM Documentation web page](#).

Remote Direct Memory Access protocol over Converged Ethernet

In addition to the OSA-Express features, IBM Z offers Remote Direct Memory Access protocol over Converged Ethernet (RoCE) features. The RoCE supports the Shared Memory Communication-Remote (SMC-Rv2) protocol, which allows operating systems to communicate through shared memory across platforms. SMC-Rv2 offers high performance, low latency network options.

As with OSA-Express, the RoCE-Express features provide 25 GbE and 10 GbE options. RoCE also can be used as a normal TCP/IP device, not just for RDMA and SMC-R. The difference is that OSA 10/25 GbE has one port per feature, while RoCE cards have two ports per feature.

Internal Shared Memory

Internal Shared Memory (ISM) is a virtual PCI network adapter that enables direct access to shared virtual memory, which provides a highly optimized network interconnect for IBM Z platform intra-communications. Shared Memory Communications-Direct Memory Access (SMC-D) uses ISM.

SMC-D v2 optimizes operating systems communications in a way that is not apparent to socket applications. It also reduces the CPU cost of TCP/IP processing in the data path, which enables highly efficient and application-transparent communications.

SMC-D v2 requires no extra physical resources (such as RoCE Express features, PCIe bandwidth, ports, I/O slots, network resources, or Ethernet switches). SMC-Dv2 does not communicate through HiperSockets OSA. Instead, SMC-D v2 performs an initial handshake over these communication paths and then, no longer uses them. Also, this handshake can occur over a RoCE TCP/IP connection.

HiperSockets

HiperSockets is another memory-to-memory communication option that is available between LPARs within the IBM Z platform. HiperSockets is an integrated firmware function that uses an internal QDIO (iQDIO) architecture to provide an efficient and secure internal network.

Because it is an internal network, HiperSockets avoids the cost of the physical network infrastructure. The Converged HiperSockets capability combines the existing high-performance attributes of HiperSocket for intra-CPC communications with your external LAN to provide a single IP network administrative model.

From a KVM perspective, these network interfaces, are available to the hosts, guests, or both. ISM and SMC-D are not available to KVM guests; only to hosts. The network interfaces can also communicate through Open Virtual Switch, MacVTap, or by using PCI Pass-through.

For more information about options and considerations, see Chapter 2, “Planning for the Kernel-based Virtual Machine host and guest” on page 21.

Cryptography

Extensive use of encryption is one of the most effective ways to help reduce potential risks and financial losses that are caused by data breaches. Encrypting data can also help meet the needs of complex compliance mandates and security best practices.

With IBM Z, the term *pervasive encryption* is used to describe the notion that all data must be encrypted, not only what is considered to be important. Pervasive encryption is enabled through tight platform integration that spans the entire IBM Z stack (hardware, software, operating systems, middleware, and even tooling).

IBM Z provides the following unique capabilities that help achieve pervasive encryption in a cost-effective way:

- ▶ On-chip crypto acceleration is performed with CP Assist for Cryptographic Function (CPACF). This hardware acceleration is provided on every processor core. It is well suited to high-speed, bulk encryption with lower latency and no CPU overhead. CPACF is included as part of the Z base system at no extra cost.
- ▶ The IBM Z platform offers a Hardware Security Module (HSM) with tamper-responding cryptographic hardware in the Crypto Express feature. The HSM protects the encryption keys. CPACF can encrypt and decrypt data by using protected keys. Protected keys are created as part of a process that includes a master key that is stored in the HSM.

The IBM Z platform uses the concept of a cryptographic domain to virtualize the physical coprocessor of the Crypto Express feature. Multiple LPARs and different operating systems can share a Crypto Express coprocessor.

IBM Z firmware enforces domain usage. The Crypto Express coprocessor manages the assignment of master keys to cryptographic domains. Cryptographic key material for one domain is not usable by another domain with a distinct master key. The Crypto Express7S meets the Federal Information Processing Standard (FIPS) 140-2 at Level 4 for cryptographic modules.

KVM supports these encryption options through standard interfaces, such as dm-crypt, which provides a transparent encryption of block devices. A passphrase or a key file is used with CPACF or by using pass-through of the Crypto Express adapter domains to the KVM guests to encrypt and decrypt the volume.

For more information about pervasive encryption for Linux on IBM Z, see *Getting Started with Linux on Z Encryption for Data At-Rest*, [SG24-8436](#).

Hardware Management Console

The Hardware Management Console (HMC) runs a set of management applications that interface with the IBM Z hardware by using a Support Element (SE) console, which acts as the single point of control. The HMC is a closed system; that is, no other applications can be installed on it.

The HMC is used to set up, manage, monitor, and operate one or more IBM Z platforms. It manages and provides support utilities for the hardware and its LPARs. The HMC also is used to add and remove processors, memory, network adapters, and storage groups to LPARs.

The HMC is used to install Linux in an LPAR on the Z platform. That Linux image can then be enabled as a KVM hypervisor.

For more information about the HMC, see IBM Support's [Hardware Management Console Operations Guide](#).

1.2 KVM working on IBM Z

KVM technology is a virtualization technology that is supported on many platforms. It turns the Linux kernel into an enterprise-class hypervisor by extending the hardware virtualization support that is built into the IBM Z platform. KVM can perform various functions, such as scheduling tasks, dispatching CPUs, managing memory, and interacting with I/O resources (storage and network) through PR/SM.

KVM on IBM Z creates VMs as Linux processes that run images by using a platform-adapted version of another open source package, which is known as a *quick emulator* (QEMU). QEMU provides I/O device emulation and device virtualization inside the VM.

The KVM kernel module provides the core for the virtualized infrastructure. It can schedule VMs on real CPUs and manage their access to real memory through PR/SM. QEMU runs in a user space and implements VMs BY using KVM module functions.

QEMU virtualizes real storage and network resources for a VM, which in turn uses *virtio* drivers to access these virtualized resources, as shown in Figure 1-2.

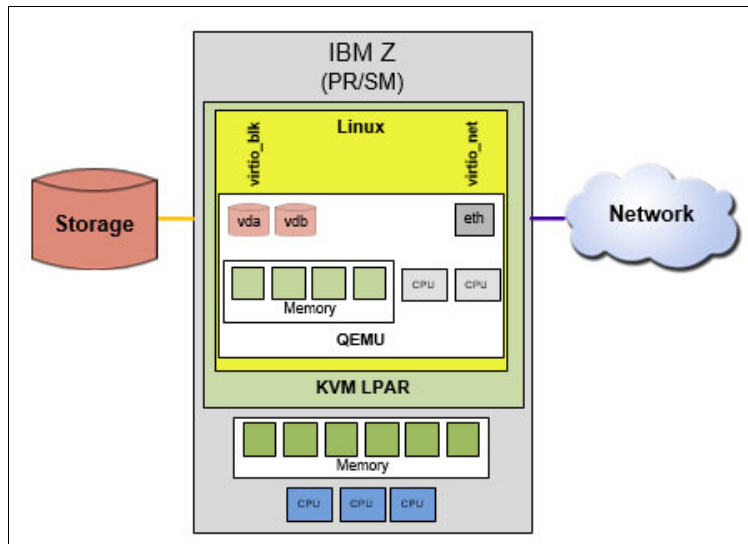


Figure 1-2 Open-source virtualization with KVM on IBM Z

QEMU also provides management and monitoring functions for VMs that are running on KVM. For more information, see the [QEMU.org wiki](#).

The network interface in Linux on IBM Z is a virtual Ethernet interface. The interface name is *eth*. Multiple Ethernet interfaces can be defined to Linux and are handled by the *virtio_net* device driver module.

Other network virtualization functions are provided by way of the following components:

- ▶ Open vSwitch (OVS), which is open source software that allows for network communication between VMs and the external networks that are hosted by the KVM hypervisor (OVS is a type of bridge). Non-OVS bridges are another networking option that is available. The KVM default network is a bridge and BONDS and VLAN subinterfaces are added constructs. For more information, see [this website](#).
- ▶ MacVTap, which is a device driver that is used to simplify virtualized bridged networks. It is based on the mcvlan device driver. MacVTap enables direct connects between a KVM host, guests, and network interfaces. It also supports Virtual Ethernet Port Aggregator (VEPA).

For more information, see this [web page](#).

In Linux on IBM Z, virtual block devices are used rather than real devices, such as ECKD or SCSI devices. The virtual block devices are handled by the *virtio_blk* (or *virtio_SCSI*) device driver module. New capabilities include pass-thru block storage.

For more information about KVM on IBM Z, see [IBM Documentation](#).

1.3 Managing and monitoring KVM on IBM Z

Effective management of a hypervisor environment requires a set of tools that address administration, monitoring, deployment, and day-to-day operations. KVM includes a standard set of tools as part of the package. These tools include drivers, APIs, system emulation support, and virtualization management.

Figure 1-3 shows a high-level view of various interfaces and tools that are available for KVM and guest VMs.

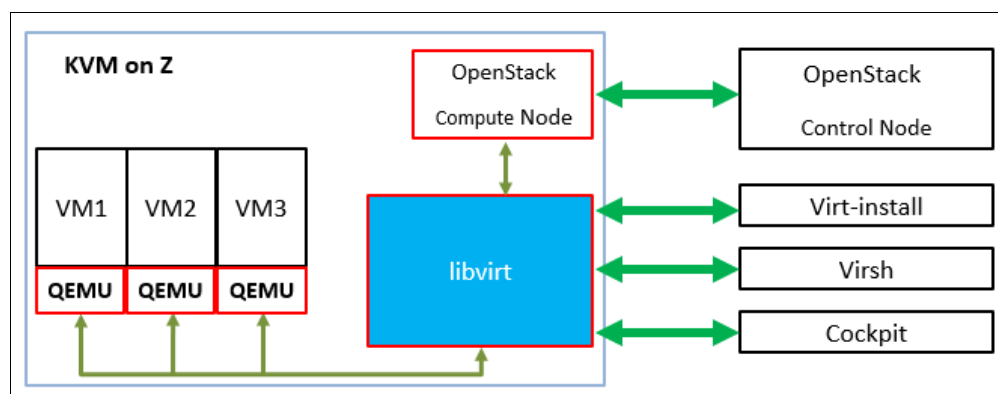


Figure 1-3 Management and monitoring interfaces

1.3.1 Libvirt

Libvirt, the virtualization API, features a common layer of abstraction and control for VMs that are deployed within many different hypervisors, including KVM. The main components of libvirt are the control daemon, a stable C language API, a corresponding set of Python language bindings, and a simple shell environment.

As of this writing, all KVM management tools (including Virt-install, virsh, and OpenStack) use libvirt as the underlying VM control mechanism. Libvirt stores information, such as the disk image and networking configuration, in an .xml file. This file is independent of the hypervisor in use.

For more information about libvirt, see the [libvirt website](#).

1.3.2 OpenStack

OpenStack is an open source framework that can manage a pool of virtualized compute, storage, and network resources through libvirt and present them to the user as a service in a secure and organized way.

OpenStack administrators can use one of the following options to manage their infrastructure:

- ▶ Command-line interface (CLI)
- ▶ Dashboard
- ▶ REST APIs

OpenStack also provides a self-service user interface that enables users to manage their own VMs, request new VMs, manage permissions, and so on.

An OpenStack environment is composed of the following node classes (see Figure 1-3 on page 10):

- ▶ The *controller node* manages the virtual resources of the compute nodes in the cloud environment. Every cloud features one controller node, and each controller node can manage more than one cloud.
- ▶ The *compute node* represents the nodes that can be managed by the controller

A node can have both the controller and compute services.

For more information about OpenStack, see the official [OpenStack Documentation](#).

1.3.3 Virt-install

Virt-install is a command-line tool for creating KVM guests and uses the libvirt hypervisor management library. You can create a VM and start an installation from the command line. A VM guest can be configured to use one or more virtual disks, network interfaces, and other devices.

Virt-install fetches the minimal files that are necessary to start the installation process, which allows the VM guest to fetch the rest of the operating system distribution, as needed. Virt-install also can run unattended for automated guest installations.

1.3.4 Virsh

Virsh provides an easy-to-use console shell interface to the libvirt library for controlling guest instances. Each of the commands that are available in virsh can be used from the virsh environment or called from a standard Linux console. Consider the following points:

- ▶ To start a virsh environment, run the virsh shell program with no options. This process opens a new console-like environment on which you can run any of the built-in commands for virsh.
- ▶ To use the virsh commands from a Linux terminal, run **virsh** followed by the command name and command options.

For more information about virsh, see Chapter 7, “High Availability for IBM General Parallel File System” on page 211.

1.3.5 Virt-manager

The virt-manager application is a desktop user interface for managing VMs through libvirt. It primarily targets KVM VMs, but also manages Xen and LXC (linux containers). It presents a summary view of running domains, and their live performance and resource utilization statistics.

Wizards enable the creation of domains, and the configuration and adjustment of a domain’s resource allocation and virtual hardware. An embedded VNC and SPICE client viewer presents a full graphical console to the guest domain.

The following Virt-Manager supporting command-line tools are available:

- ▶ **virt-install**: Provides an easy way to provision operating systems into VMs.
- ▶ **virt-clone**: Used cloning inactive guests. It copies the disk images, and defines a configuration with new name, UUID, and MAC address that points to the copied disks.
- ▶ **virt-xml**: Used for easily editing libvirt domain XML by using virt-installation command line options.
- ▶ **virt-bootstrap**: Provides an easy way to set up the root file system for libvirt-based containers.

Figure 1-4 shows the virsh interface with libvirt for virtual server management.

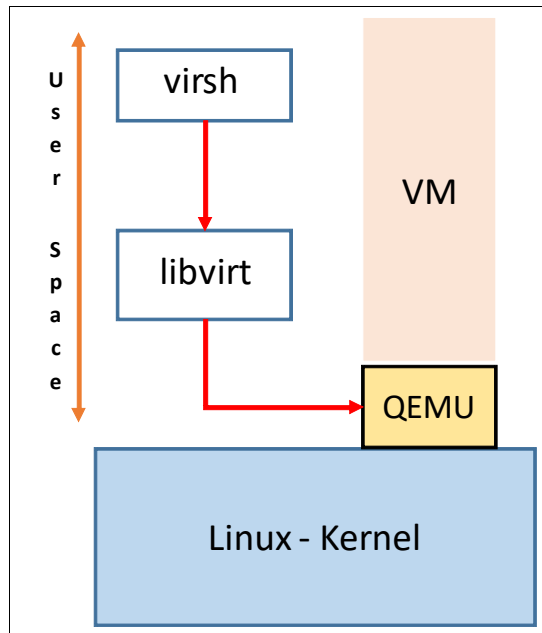


Figure 1-4 Libvirt interface with virsh

1.3.6 Cockpit

Cockpit is an open source project that provides a web browser interface to manage many system functions, including the KVM hosts and guests management. It simplifies the management of large environments and collects data through standard APIs and components, such as QEMU and Libvirt (see Figure 1-5 on page 14).

Cockpit is one of many possible tools for managing and monitoring VM hosts and guests. The cockpit is an excellent alternative to commercial software. It can be customized according to the business needs and evolves with the new versions.

For more information about cockpit features, see 6.1.3, “Cockpit” on page 192.

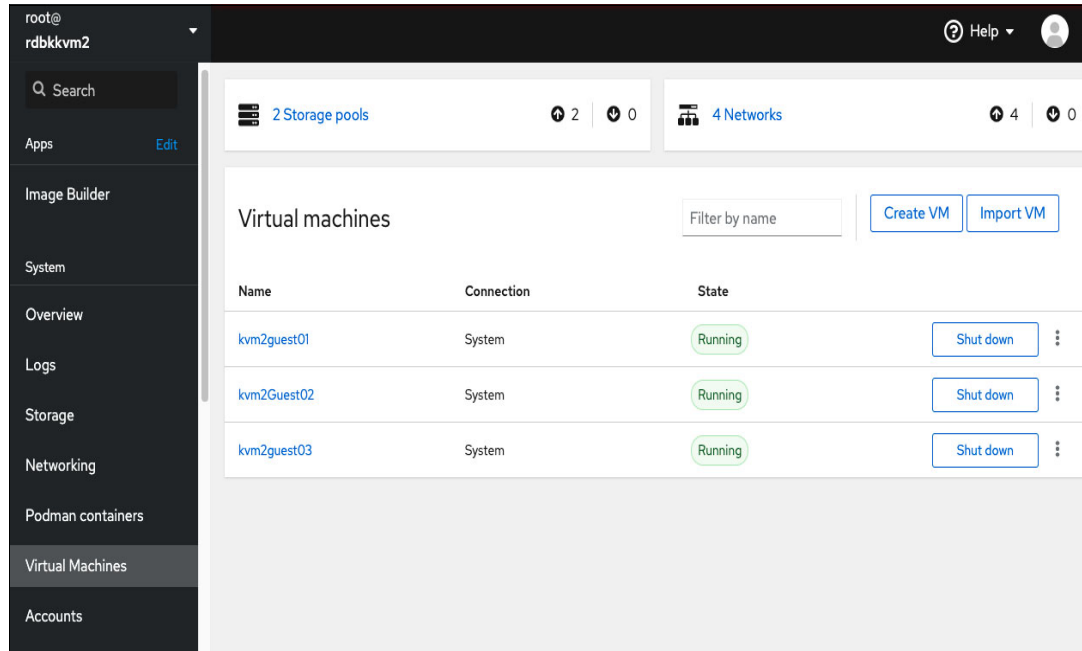


Figure 1-5 Cockpit VM Module

1.3.7 IBM Cloud Infrastructure Center

The IBM Cloud Infrastructure Center is an Infrastructure as a Service (IaaS) solution that provides an industry-standard user experience for cloud infrastructure management. It provides KVM-based virtual infrastructure management and allows for the automation of services.

The IBM Cloud Infrastructure Center enables the integration between different areas of the enterprise, which simplifies the management of the virtual environment. As a result, costs in the IT infrastructure are optimized.

Also, it provides the capability to integrate IBM Z® and IBM LinuxONE into a hybrid cloud model across the enterprise and a foundation for a scalable infrastructure cloud management solution. ICC includes the following key features:

- ▶ Industry-standard user experience
- ▶ Efficient infrastructure management and provisioning
- ▶ Image provisioning for fast deployments
- ▶ Self-service portal for users
- ▶ Integration by using OpenStack compatible APIs
- ▶ Accelerate cloud deployments

For more information about IBM Cloud Infrastructure Center capabilities, see Chapter 9, “IBM Cloud Infrastructure Center on Kernel-based Virtual Machines” on page 321.

1.3.8 Platform management

Various levels of management exist with KVM on IBM Z, and the function is built into different levels. The management functions include the following examples:

- ▶ Unattended installation of the KVM hypervisor simplifies administration. The mechanism for automation varies depending on the Linux distribution:
 - Kickstart with Red Hat
 - AutoYast with SUSE
 - Preseed with Ubuntu
- ▶ Hypervisor management GUI tools
Open-source management tools that provide an intuitive graphical user interface (GUI) for the following host configuration management tasks:
 - Networking configuration for RoCE and OSA-based NICs
 - Storage configuration for ECKD, SCSI devices, Network based storage and NVMe
 - System basic information and statistics
 - Debug reports
 - Hypervisor shutdown/restart

1.3.9 Managing the KVM guest lifecycle

Several tools are available to manage the various lifecycle functions of the VM guests, including the following examples:

- ▶ IBM Cloud Infrastructure Center
- ▶ VM Manager
- ▶ oVirt
- ▶ OpenStack
- ▶ Cockpit
- ▶ virsh CLI
- ▶ QEMU guest agent

Some of the tasks that are handled by these tools include start and shutdown, cloning a guest, removing a guest, taking snapshots (point-in-time copies) of a guest, saving and restoring, and suspending and resuming a guest.

1.3.10 KVM host and guest monitoring

Monitoring availability, utilization, and performance are an important part of the daily operation of the virtualized environment. Open source and commercial solutions use standard tools and interfaces to achieve this task.

Tools, such as Nagios and Icinga, can perform various checks, while analytic tools, such as Elastic Search, Logstash, Kibana (ELK), Graphite, Grafana, Prometheus, and Collectd, provide visualization of monitoring and performance data. Many other tools (such as systat, virt-top, and sar) are available for monitoring, most of which can monitor the KVM host and the guest.

1.4 Securing KVM on IBM Z

IBM Z often is described as being a highly secure platform. Security is built into the foundation of the IBM Z platform, but it still requires careful planning and configuration.

Extra effort is required to protect your business data by planning and enabling the system's features. The necessity of this effort is especially true in a virtualized environment, such as KVM on IBM Z where thousands of guests and their data can be at risk, and poor security practices might be easily replicated throughout an environment.

1.4.1 Access control

Security through access control is most often thought of in Linux as the ability to control read, write, and run for owners, groups, and everyone else. Also, access control lists can be used with the Linux commands: Set File Access Control Lists (`setfacl`) and Get File Access Control Lists (`getfacl`).

Red Hat Identity Manager (IdM) provides an integrated identity management service for Linux⁴. At its core, IdM combines LDAP, Kerberos, DNS, and PKI with a rich management framework. Frequently, IdM is described as “Active Directory for Linux”.

For more information, see this [Red Hat web page](#).

Other levels of security are possible with various Linux kernel security modules and components, such as AppArmor, Security-Enhanced Linux (SELinux), Polkit (PolicyKit), and Linux Pluggable Authentication Modules (PAMs). These extra levels of security provide for the use of profiles and policies that can control access and usage for different resources.

1.4.2 IBM Secure Execution on IBM z15 and newer IBM Z systems

IBM Secure Execution for Linux is a technology that was introduced with IBM z15 and LinuxONE III. It was created to protect the data of workloads that run in a KVM guest from being inspected or modified by the host environment.

Specifically, no hardware administrator, KVM code, or KVM administrator can access the data in a guest that was started as an IBM Secure Execution guest.

Therefore, IBM Secure Execution for Linux continues and expands well-known security features of IBM Z and LinuxONE. It supplements pervasive encryption, which protects data at rest and data in-flight and protects data in-use. With IBM Secure Execution for Linux, it is possible to securely deploy workloads in the cloud. The data of the workload can be protected everywhere, as shown in the following examples:

- ▶ In flight with secure network protocols, such as TLS, SSH, or IPsec. At rest, with volume encryption (such as dm-crypt) or file system encryption (such as with IBM Spectrum Scale).
- ▶ In use in the memory of a running guest with IBM Secure Execution protection when a KVM guest runs in a cloud, whether it is in-house or third-party, security risks to the workload include:
 - Intruders who might gain root privileges because of some error in the security administration of the hypervisor.

⁴ IdM provides an integrated identity management for various platforms, including MAC and Windows.

- Malicious hypervisor code that might be introduced by users, including zero-day users or intruders.
- Malicious VMs that, hypothetically, can escape the control of the hypervisor, and gain hypervisor privileges.
- Intruders, malicious hypervisors, or malicious VMs are risks for the cloud provider and the cloud customer.

To provide a secure hosting environment, a cloud provider might log every key stroke and conduct expensive audits to log any management action and deter any malicious actor.

With IBM Secure Execution, data is protected during processing. As a workload owner, your data in your KVM guest that is deployed in a cloud (which runs on IBM Z servers with IBM Secure Execution) are as safe as though you run it in your own data center. In fact, it is safer. It is also protected from insider attacks. Only the workload owner can access the data.

Benefits of IBM Secure Execution

IBM Secure Execution provides the following benefits:

- ▶ Instead of relying on deterrence by using extensive audit tracks, IBM Secure Execution provides technology-enforced security rather than process or audit-based security.
- ▶ As a cloud provider that uses IBM Secure Execution, you can attract sensitive workloads that, used to be restricted to the workload owner's system.
- ▶ As a secure workload owner, you know that your workload is run securely, even outside your data center.
- ▶ As a secure workload owner, you can choose where to run your workload, independently of the required security level.

1.4.3 Authentication solutions

Solutions for authentication range from basic, built-in solutions to more complex open source projects to commercial solutions.

At the most basic level, `/etc/passwd` can be used for authentication, but this solution does not scale well in a complex, multi-server environment.

Lightweight Directory Access Protocol (LDAP) is another option, which provides more capabilities and better scalability. However, administration of LDAP can be a challenge.

The Open Source project FreeIPA provides centralized authentication, authorization, and account information by storing data about users, groups, hosts, and other objects that are necessary to manage the security aspects of a network of computers. FreeIPA is built on top of well-known, open-source components and standard protocols with a strong focus on ease of management and automation of installation and configuration tasks.

Commercial authentication solution, such as Red Hat IdM (see 1.4.1, "Access control" on page 16) also are available. In some cases, these solutions can provide the added value of integrating security policies across heterogeneous platforms.

1.4.4 Multi-factor authentication

Increased security requirements and regulations are leading businesses to move toward multi-factor authentication (MFA) solutions. These solutions are based on the One Time Password (OTP) standard, which includes HOTP and TOTP.

HOTP solutions use Hash-based Message Authentication Code (HMAC) OTP to generate a password and is valid based on an event. TOTP solutions use Time-based OTP, which provides a password that continually changes based on the time since it was generated. In both cases, the passwords are valid only for a short time.

Open Source and commercial options are available for implementing MFA. Some examples are [Red Hat IdM](#) and [Google Authenticator](#).

1.4.5 Audit

An important aspect of security is the ability to audit the system activity. The Linux Audit package provides auditing capabilities, including providing sample policies. Commercial solutions also are available, such as IBM QRadar, which adds value through the log data analysis.

1.5 Availability with KVM on IBM Z

Availability on the IBM Z platform begins at the lowest level with the reliability of the hardware. IBM Z platforms feature a mean time between failure (MTBF) that is measured in decades, with redundancy that is built-in, which makes services and application availability 99.999% possible. As business requirements demand higher levels of availability, it is important to plan for and choose options that support these requirements.

Looking outside the KVM hypervisor, decisions (such as configuring more than one network and storage interface) must be considered to avoid a single point of failure. Planning ahead to support disk subsystem changes can help avoid downtime.

Running KVM on IBM Z adds to the value of the platform. For planned events, such as maintenance, KVM provides for live guest migration that allows for a VM guest to be moved between KVM hosts while workloads are running. This function is built into KVM, and the correct configuration is required to avoid issues.

For unplanned events, relying on the resiliency capabilities of the middleware or application is a good approach. The ability to recover from a failure is based on the capabilities of each individual application, middleware, and program to determine the following information:

- ▶ How a failure can be handled and how a failure can be avoided when a failure occurs.
- ▶ The severity of a failure to its operation and whether processing must be stopped.
- ▶ What data is required to expedite failure diagnosis.

Also, workloads that are running in a VM guest can have their own high availability solution. Middleware products from IBM, such as Db2, IBM MQ, and IBM WebSphere all provide their own recovery and clustering technology to support high availability.

Commercial and open source solutions also are available. For example, IBM System Automation can automate the restart of a failed guest or movement of workload to another guest if a failure occurs.

The Linux-HA open source project provides clustering solutions that are based on standard Linux packages (that is, Corosync and pacemaker) and cluster file system solutions (such as IBM Spectrum Scale) to handle unplanned events.

For more information about considerations for planned and unplanned outages, see Chapter 2, “Planning for the Kernel-based Virtual Machine host and guest” on page 21.

1.6 KVM on IBM Z backup and recovery

Backup and recovery of KVM hosts and guests are important to the overall operations of the environment. They must be planned for and more importantly, tested. File backups and volume backups are conducted at a high level.

This process includes backing up the various KVM hypervisor components, such as the operating system disk, storage that is used for host image files, system logs, and key configuration files. At the VM guest level, the options can vary based on the workload that is running in the guest. For example, a database that is running in the guest might backup and recovery utilities that operate independent of the guest.

At the most basic level, full volume backup solutions, such as IBM FlashCopy, are available for IBM storage technology. IBM FlashCopy provides a point-in-time copy that works with ECKD and SCSI devices. In this case, a disk is copied at the block level to another disk or tape. Other commercial (that is, IBM Spectrum Protect TSM) and open source solutions (that is, Amanda, Bacula, and rsync) are available, which provide for volume and file-level backup and recovery.

For file system and databases backups, several open source and commercial options are available. Choosing the best solution is part of the architecture study and depend on many factors, including software features, team skills, and software licensing costs. As always, you know the best backup solution for your environment.

Also, disk devices from other vendors provide disk-level backup options. The many options must be considered during the planning phase for implementing KVM.

Backup and recovery can be included as part of a large disaster recovery strategy, which also can consist of replicated disk options. Again, backup and recovery and disaster recovery are key to any IT operation that reaches beyond a KVM on IBM Z discussion.

To support many characteristics that provide redundancy for Disaster Recovery and storage backup, the IBM GDPS implements solution clustering technology for IBM Z, which includes server and storage replication and automation⁵.

⁵ GDPS does not provide much value in the Linux space; it is primarily designed for z/OS. Storage replication in Linux environments can often occur without GDPS.

IBM GDPS is a collection of system recovery offerings on the IBM Z platform, each of which uses services, clustering technologies, and server and storage replication and automation. This technology features the following benefits:

- ▶ Resiliency to support multiple disk replication architectures, high availability, and disaster recovery at any distance from a single point of control in seconds.
- ▶ Flexibility to gain data consistency across Linux hosts and guests. GDPS also uses open architecture and is application-independent.
- ▶ Automation of operational tasks for availability and recovery with scripts.
- ▶ Monitoring of your environment from a single point and in real time to detect any deviation that might affect resiliency.

For more information about IBM GDPS, see *IBM GDPS: An Introduction to Concepts and Capabilities*, [SG24-6374](#).

For more information about backup and recovery and IBM GDPS in the KVM environment, see Chapter 2, “Planning for the Kernel-based Virtual Machine host and guest” on page 21.



Planning for the Kernel-based Virtual Machine host and guest

When building an IT infrastructure with the highest quality of service, comprehensive planning is critical to long-term success. Aspects that you might not consider when first deploying hardware and software resources can have a significant effect on the IT infrastructure later on.

For example, when you plan the deployment of Kernel-based Virtual Machine (KVM) hosts and guests, anticipate adding and replacing resources as environments that typically evolve over time. This planning might include migrating nondisruptively to a new storage server.

How you initially provision your storage can affect your ability to perform such a nondisruptive migration.

Similarly, successful live guest relocation depends on device addressing and naming conventions. If you configure device names and addresses without careful thought, that process can translate to a different device in another KVM host, which can cause the guest relocation to fail.

This chapter provides guidance and best practices for KVM hosts and guests, including guidance for planning deployment, usage of hardware resources, and use of IBM Z capabilities. Considerations for management and monitoring, back up and recovery, availability, and security are also included.

This chapter includes the following topics:

- ▶ 2.1, “Basic requirements for KVM hosts and guests” on page 22
- ▶ 2.2, “Planning resources for KVM guests” on page 26
- ▶ 2.3, “Planning for management and monitoring” on page 37
- ▶ 2.4, “Planning for security” on page 40
- ▶ 2.5, “Planning for backup and recovery” on page 43

2.1 Basic requirements for KVM hosts and guests

Requirements are ultimately driven by business needs. These needs determine the availability, scalability, and function essentials of your IT environment. Business requirements also influence how the IBM Z platform, storage resources, network resources, and connectivity are designed and configured to deliver the expected levels and quality of service.

2.1.1 Hardware requirements

KVM guest hardware requirements are the same as the KVM host. For more information about these requirements, see the [Linux on IBM Z Tested Platforms](#) web page and your Linux distributor's installations documentation.

System capacity requirements are based on the size and number of guests that you plan to host. These requirements include the prerequisites of the middleware and applications that you chose to run in them. It is important to define the logical partition (LPAR), KVM host, and KVM guests in a manner that allows you to dynamically add and remove resources as needed.

Basic hardware requirements include at least one virtual CPU, enough processor memory to install or start an image, enough disk storage to store at least the operating system and optionally a network device. The KVM host might use general-purpose central processors (CPs) or Integrated Facility for Linux (IFLs) processors.

Note: Simultaneous multithreading (SMT) is available on IFLs only. When SMT is used, remember that a virtual processor is a thread and not a full core CPU.

A KVM guest includes the following minimum requirements:

- ▶ One or more virtual CPUs (an IFL or a CP)
- ▶ A total of 512 MB - 2 GB of memory (the Linux installer might require more than the minimum)
- ▶ A total of 500 MB - 1 GB of storage per image (20 GB is more of an ideal starting point)

The storage disk typically is an image file on the KVM host or a block device that is defined to the guest. Check with your storage vendor for support with IBM Z hardware. For more information about IBM Storage, see the [IBM Systems Storage Interoperation Center \(SSIC\) web page](#).

No network interface is needed for installing a guest image. If network connectivity is required for your environment, it can be achieved by using one of the following methods:

- ▶ The KVM "Default Network" that uses KVM host connectivity. This selection is the most commonly used selection during installation.
- ▶ A RoCE Express feature that uses the SMC-R protocol. The connection is made by way of PCI pass-through.
- ▶ An OSA-Express feature. Each interface (port) can be shared through an Open Virtual Switch or MacVTap by using the virtio driver in the guest.
- ▶ A HiperSockets LAN. The virtual interface can be accessed by the guest by way of MacVTap by using the virtio driver.

Note: As of this writing, the Internal Shared Memory (ISM) interface that uses the SMC-D protocol is *not* supported by KVM guests. However, ISM is supported in KVM hosts.

2.1.2 Software requirements

A basic installation (or use) of Linux on IBM Z as a KVM guest does not include any special requirements. What is included in the Linux distributions is fully ready to be virtualized. Drivers, such as virtio, are included and automatically used at installation time.

The [Linux on IBM Z Tested Platforms](#) web page maintains information about vendor certifications and hardware levels. It also provides more information about specific kernel levels that are required for compatibility with new hardware. In addition to including the certified levels for general operations, it includes support for cryptography.

Refer to your Linux distribution for more information about requirements that pertain to the following installations of your guest Linux images:

- ▶ [Red Hat](#)
- ▶ [SUSE](#)
- ▶ [Ubuntu](#)

To add enhancements or add functions, packages might be available that must be added to guests, including the following examples:

- ▶ OpenStack: In this environment, you want to be sure to have cloud-init installed in the image that is to be deployed.
- ▶ QEMU guest agent: Adds interoperability between KVM hosts and guests. It allows the KVM host to query and manage resources in the guest operating system. Use cases can include taking offline a virtual CPU in a guest or freezing a guest file system for a Snapshot.
- ▶ Monitoring agents: You might want to include agents, such as ITM, Prometheus, Collectd, or Nagios, to gather usage and availability data. You might also want to ensure performance data collection from tools, such as systat and sar, are in place.
- ▶ Management agents: Many tools use the KVM host libvirt interfaces to manage or operate a guest. Tools, such as Virtualization Manager, use these interfaces. Other management tools, such as CockPit, can be in the KVM host and KVM guest, each listening on their own TCP port.

2.1.3 Availability requirements

Critical IT components *must* have available backup (*redundant*) capacity, power sources, and connections across critical paths to storage, networks, and other systems, and multiple instances of software.

However, redundancy alone does not necessarily provide higher availability. It is essential to design and configure your IT infrastructure by using techniques technology that can use redundancy and respond to failures with minimal effect on service availability.

For guest virtual machines (VMs) that require enhanced availability beyond the availability that is provided by a single guest in a single KVM host, the following events should be considered:

- ▶ **Planned events:** Live guest migration is an option to move a guest from a KVM host in one partition to another KVM host on a different partition or IBM Z platform. Network connectivity is required. The use of live guest migration can have implications in how you design your network connectivity and the names and addresses that are used to reference block device storage.
- ▶ **Unplanned events:** If you require high availability (HA) for unplanned events, review the middleware or application that the VM hosts. Many middleware programs feature their own HA capabilities. These capabilities generally are used first because they might be tuned for the application or include some application-specific tailoring ability, testing and certifications, or specific environment requirements.

For example, Oracle RAC, IBM Db2 HADR, IBM WebSphere Application Server cluster, and even IBM MQ all include their own HA features and requirements. For middleware or applications that lack built-in HA, consider Open Source Linux HA (for a commercial solution, consider IBM Systems Automation).

Linux HA uses several different packages, including pacemaker, corosync, drbd, and clvm. You can use clustered or shared file systems, such as Spectrum Scale (gpfs), gfs, ocfs2, ceph, glusterfs, and nfs. Linux HA also can work with KVM/libvirt to move KVM guests to other KVM host instances.

- ▶ **Storage server moves:** Consider the use of LVM at all layers and the **pvmove** command to dynamically exchange individual volumes. Another possible method can be to use software RAID mirroring to move volumes from one storage server to another.
- ▶ **Network availability and redundancy:** Use network interface bonding or link aggregation. Both options are the most common methods to add enhanced availability and capacity to network connectivity.

Link aggregation requires OSA-Express ports to be dedicated to a partition (not dedicated in HCD). Dedicating multiple OSA-Express ports to a single partition is *not* an efficient use of resources.

Bonding can provide a sort of similar availability enhancement without the restriction of dedicating an OSA-Express port to a single LPAR. KVM guest connectivity generally requires more availability and capacity than the KVM host.

The HMC ASCII console can always be used as a backup method to administer the KVM host if a temporary network connectivity loss occurs.

- ▶ **Storage server connectivity:** It is standard practice to always include multiple Fibre Channel connections from the IBM Z connected to a minimum of two SAN fabrics. A minimum of two Fibre Channel ports also can be used on your storage server.
- ▶ **Securing data:** To perform pervasive encryption and use protected and secure keys a pair of Crypto Express features must be used, at a minimum. Crypto Express domains can be configured to only a single KVM guest.

Because a maximum of 85 domains per Crypto Express adapter is available, plan for a pair of features for every 85 KVM guests. If the KVM host uses encryption, you must account for the extra domain usage when planning. The CPACF is a part of every processor and generally no configuration is required beyond the enabling microcode feature.

- ▶ Internal NVMe storage availability: Although this SSD storage is internal and has few parts that can fail, the storage must be configured in RAID configuration if a solid-state drive (SSD) fails.
- ▶ Compression/decompression acceleration: The accelerator unit for these operations is part of every processor. No special guest configuration is required for use beyond being at the required Linux software levels.

The following Data Compression with Linux on IBM z15 videos are now available on the IBM Media Center Channel:

- [Accelerated Data Compression with Linux on IBM z15 - Managing Data Growth](#)

Manage your data growth by reducing your storage requirements and your data transfer by up to 80% without changing your I/O infrastructure.

- [Accelerated Data Compression for Linux Databases on IBM z15](#)

Learn how you can boost the performance of your Linux databases by using accelerated data compression with Linux on LinuxONE III.

2.1.4 Deployment architecture

From your requirements, you arrive at a deployment architecture. No one architecture is for every use because requirements are unique in each instance.

Major influencers to the deployment architecture might be software licensing or support, availability and scalability requirements, architectural limitations, and security requirements.

In addition, naming and addressing resources, such as network interfaces and block storage devices, also are important if live migration is used. Naming and addressing conventions must be well-defined and implemented into the deployment architecture.

For the purposes of this IBM Redbooks publication, we use a simple configuration with basic redundancy to show how the key IBM Z capabilities are deployed.

Our deployment architecture is shown in Figure 2-1.

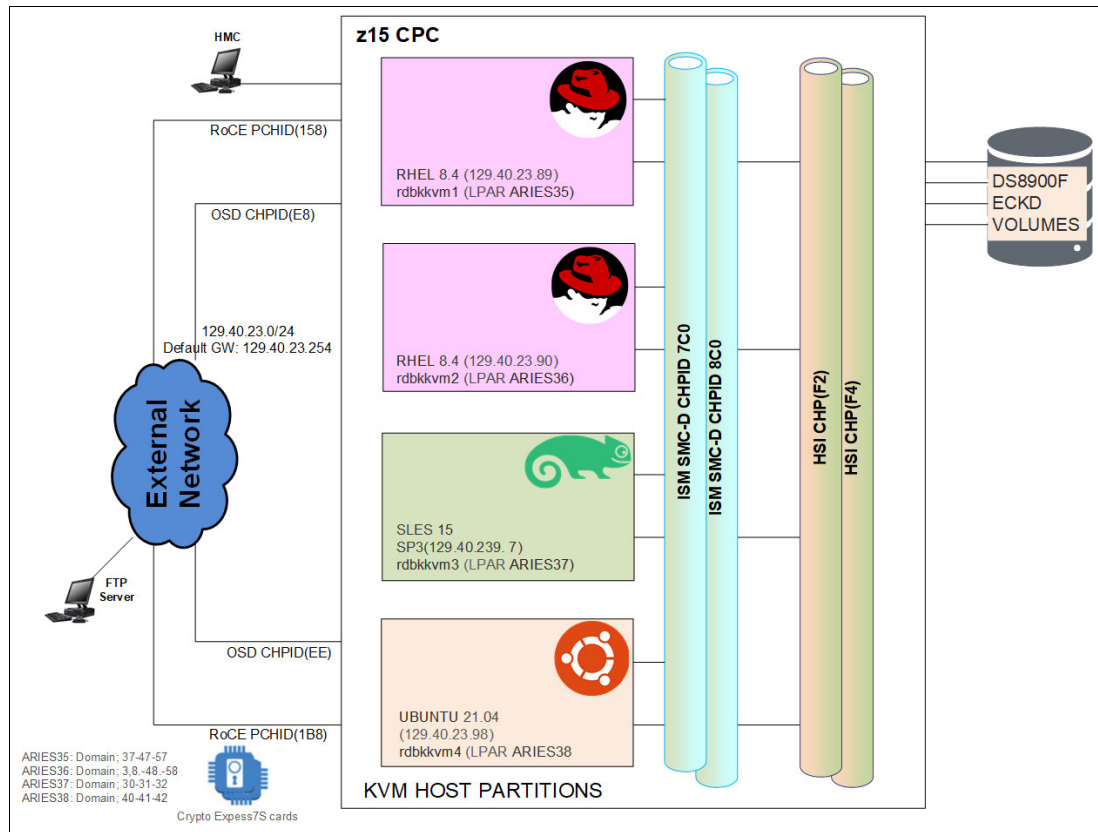


Figure 2-1 Partition infrastructure of deployment architecture

2.2 Planning resources for KVM guests

Each KVM guest requires CPU, memory, storage, and network resources. Other devices and issues you might not immediately think about are important, such as the following examples:

- ▶ Text console device
- ▶ Graphical console
- ▶ Cryptographic resources
- ▶ Virtual CDs and DVDs
- ▶ Watchdog devices
- ▶ Start details (device order, interactive menu, timeout, and kernel command-line data)
- ▶ Crash actions
- ▶ Time zone data

When you make the initial resource allocation, it is important to recognize that resource needs change over time. Where possible, make definitions in a manner so that you can dynamically adjust allocations of your resources.

It is also important to understand how you can prioritize or cap your KVM guest CPU allocations. Adding or removing disk storage from a guest domain can be a common request. Consistent and unique naming and addressing of these resources is key for live guest migration purposes.

2.2.1 Compute considerations

The basic compute resource aspect is straightforward. You can and should define an initial and maximum number of virtual CPUs for the guest domain. By default, you are running in SMT mode with IFLs. Each virtual CPU in SMT mode is half of a core and *not* a full core.

You can set relative shares by editing the VM domain definition in XML format by using the following tag:

```
<shares>2048</shares>
```

You can tailor the relative share CPU between the different guests. You also can use the period (interval in microseconds) and quota (bandwidth in microseconds) tags (`<period>1000000</period>`, `<quota>1000</quota>`) to provide a cap on CPU consumption for a guest domain.

KVM supports CPU and memory over-commitment. To maximize performance, define the minimum number of virtual CPUs and memory that are necessary for each guest. If you allocate more virtual CPUs to the guests than are needed, the system works, but a level of overhead occurs in doing so.

Consider the following preferred practices:

- ▶ CPU: Do not define more virtual CPUs to a guest than the number of logical PUs or threads that are assigned to the KVM host.
- ▶ Memory: Try to avoid an over-commit ratio of memory of more than 2:1 (virtual:real). Any paging operations are slower than processor cache or main memory.

From a performance perspective, one other aspect you must evaluate tailoring is the I/O threads. Rather than a single thread in the QEMU event loop, you can allocate multiple threads and assign different threads to different I/O devices. Make this allocation if I/O performance was of concern and multiple virtual CPUs were assigned to the guest.

Consider the following rules for I/O threads usage:

- ▶ The number of I/O threads must not exceed the number of host CPUs.
- ▶ Over-provisioning of I/O threads must be avoided. A good starting point is to have one I/O thread for every two to three virtual disks.
- ▶ Even a single I/O thread instantly improves the overall I/O performance compared to default behavior; therefore, it always must be configured.

2.2.2 Storage considerations

Linux on IBM Z and KVM can work with various storage types. The two most common types that are supported on this platform are FCP attached SCSI LUNs or Extended Count Key Data (ECKD) devices. Various network-based storage access methods also are available, but the operating system typically is installed on either of these two types.

Which to use? Both types of storage can get the job done, but it might be best to stick with what you know whether you use one type and know how to administer that type. You might have backup and recovery tools that work with that type of storage.

If you require high-performing I/O, such as for a database, we tend to see FCP-attached SCSI LUNs used more. They can achieve slightly better performance. ECKD storage can obtain similar performance if Parallel Access Volumes (PAV) are used. PAVs do require another level of administration that might not always be available.

Linux on IBM Z and KVM does support the use of multiple channel subsystems and different subchannel sets. This concern is most often for clients with many smaller ECKD devices and might even be part of a disaster recovery solution to use different subchannel sets at different sites.

A common question is: How many Fibre Channel paths are required to the storage server?

At a minimum, two paths are needed for a basic level of redundancy. More paths are possible based on I/O performance or capacity requirements.

Note: It is important to follow the best practice of *single initiator zoning*. Among other reasons for the practice are that it helps to keep the number of logical paths to a minimum. An excessive number of logical paths can cause Linux installation failures, delays during start, and when working with storage management tools.

Several options are available beyond what the KVM host can use to what is presented to a KVM guest. Ignoring network-attached storage again for the moment, block devices and image files are available.

Image files are typically qcow2 files. These images files support snapshotting. As the name implies, they provide Copy On Write support. For example, you can clone a base image to some number of new virtual servers, and each new server records only the “deltas” in a unique file.

The qcow2 files also are sparse files. You designate a maximum size, but the space is not used until they are written to. The qcow2 files can be compressed, and they also support being encrypted.

Because the image file is in the file system of the KVM host, they often do not facilitate live migration. The use of a shared file system with another KVM host can facilitate it. Technically, KVM migration support can copy an image file over the network; however, this ability is rarely practical for servers, such as large database servers.

The following example is a sample XML for raw type image file guest definition:

```
<disk type='file' device='disk' cache="none" iothread="2" >
<driver name='qemu' type='raw'/>
<source file='/var/lib/libvirt/images/guest1-0Sdisk.img' />
<backingStore/>
<target dev='vda' bus='virtio' />
</disk>
```

The following example is a sample XML for qcow2 type image file guest definition:

```
<disk type='file' device='disk' cache="none" iothread="3" >
<driver name='qemu' type='qcow2' />
<source file='/var/lib/libvirt/images/guest1-0Sdisk.qcow2' />
<target dev='vda' bus='virtio' />
</disk>
```

Note: The source file suffix for a qcow2 image file is *not* required to be qcow2. This issue can cause confusion if the administrator does not name the files consistently.

Block devices end up being /dev files in Linux that represent a real device in a device-independent manner. Block devices tend to achieve lower-latency and higher throughput than image files because they minimize the number of software layers through which it passes.

Figure 2-2 shows the SCSI and ECKD options for KVM guest.

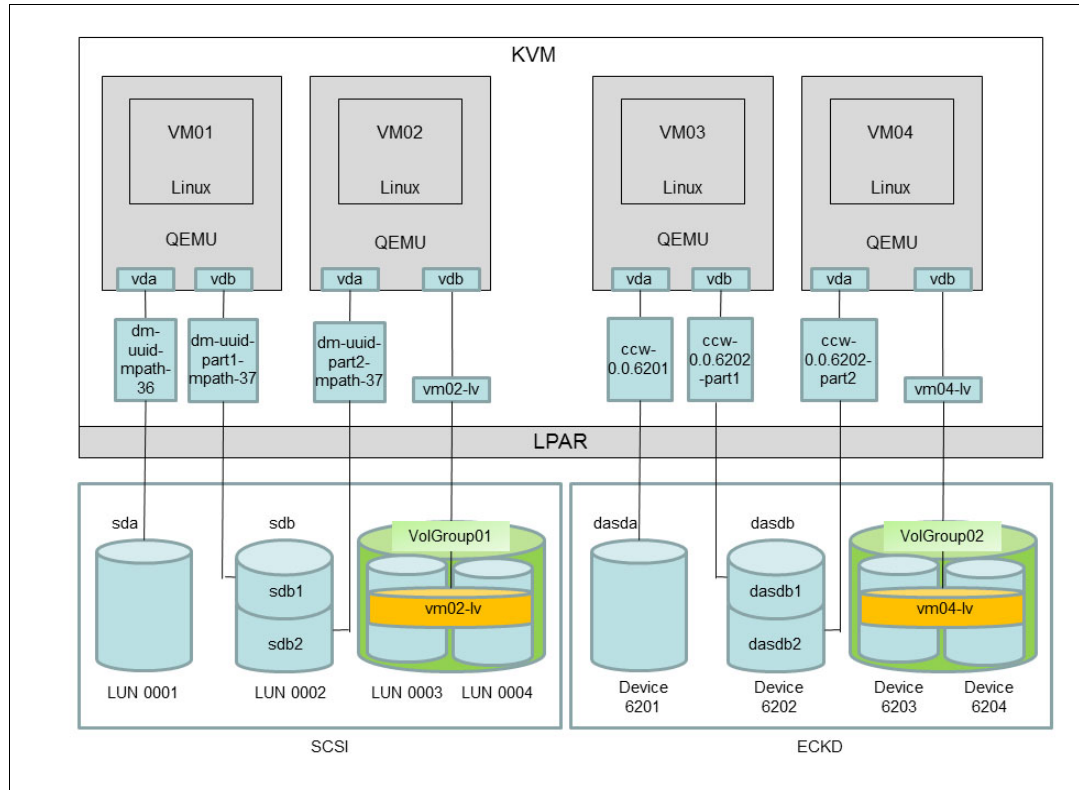


Figure 2-2 Block devices for KVM VMs

When you read/write to the /dev file for a block device, you are reading or writing to the device. From these files, you can partition disk storage, make them part of volume groups and logical volumes, place file systems on them, or in some instances reference them as raw devices without a file system.

For a KVM guest, you can partition an entire disk or only a portion of a disk and supply that partition as a block device to the guest. The use of a block device that is shared between two KVM hosts tends to be the simplest way to support live migrations. The data on disk does not need to be copied between the two KVM hosts. It is read/written directly on the storage server.

The one aspect to focus on is the name that you use to reference the shared block device. Ensure that you use a name that consistently represents the same device. Also, this name must be multipathed, which often means that the name includes the UUID of the device.

The following example is a sample XML for full block device in guest definition:

```
<disk type='block' device='disk' cache="none" io="native" iothread="2">
<driver name='qemu' type='raw'/>
<source dev='/dev/disk/by-id/dm-uuid-mpath-36005076307ffd1220000000000004203' />
<target dev='vda' bus='virtio' />
</disk>
```

The following example is a sample XML for partition of block device in guest definition:

```
<disk type='block' device='disk' cache="none" io="native" iothread="3">
<driver name='qemu' type='raw' />
<source
dev='/dev/disk/by-id/dm-uuid-part1-mpath-36005076307ffd1220000000000004202' />
<target dev='vdb' bus='virtio' />
</disk>
```

You also can create LVM-based storage pools for the guest domains to use. The benefit of LVM-based solutions is that they provide a layer of abstraction between the device and guest resource definition. LVMs typically allow you to add, move, and remove the underlying devices.

The following example is a sample XML for LVM-based block device in guest definition:

```
<disk type='block' device='disk' cache="none" io="native" iothread="1">
<driver name='qemu' type='raw' />
<source dev='/dev/Vo1Group00/LogVo100' />
<target dev='vda' bus='virtio' />
</disk>
```

Image files and block devices can be dynamically added. They also can be dynamically removed if they are no longer used by the guest.

Note: We recommend the use of multipath volumes. However, the KVM guest never sees this multipathing because it is addressed at the hypervisor layer.

One key planning aspect is to never have LVM or volume group names between the KVM host and KVM guest that are duplicated. The KVM host administrator might not control what LVM names are used by the guests.

The guest administrator can add a volume group or logical volume name that can collide with the host. These naming collisions are not visible to the guest because the guest sees only its own storage and not the storage of the KVM host.

Two approaches are available to tell the KVM host, Logical Volume Manager to ignore the KVM guest logical volumes and volume groups:

- ▶ Use the filter keyword in the `lvm.conf` to accept or reject specific names. Allow only the names that are used by the KVM host to be processed.
- ▶ Use “system id” with LVM. By using this approach, you assign a system ID name to a volume group. Also, you specify the `system_id_source` name in the `lvm.conf`. In this way, all volume groups without a matching system ID are ignored.

Image files and block devices are typically accessed by way of the virtio device driver in the guest. Pass-through support also is available for ECKD block devices.

Virtual CD-ROMs are available. You reference a `.iso` image file on the host and they are presented to the guest as a CD-ROM. On s390, the CD-ROM appear as `virtio-ccw` to the guest and contains the following components:

- ▶ **Host file systems:** If you use image files, consider the use of LVM or a shared/clustered file system for `/var/lib/libvirt` images. LVM offers storage flexibility. You can add, remove, or migrate out storage that is not apparent to the guest. The shared/clustered file system often features the same ability, but also can allow for sharing and live migration. Some file systems also allow for compression and encryption.
- ▶ **Guest file systems:** The guest file system can be whatever your Linux distribution supports.

Beyond image files, block devices, and virtual CD-ROMs, consider the following other storage resource types that are provided to a KVM guest:

- ▶ `dir`: A host directory to use as a pool for guest image files
- ▶ `netfs`: Using CIFS, NFS, gluster-based storage
- ▶ `logical`: A pool that is based on LVM volume groups
- ▶ `gluster`: A pool that is based on native gluster access to storage
- ▶ `iscsi`: Provides a pool based on an `iscsi` target

Storage server migration considerations

Planning ahead for storage server migration is a must. Storage servers never exist forever and are often replaced with new technology or perhaps to a new storage vendor. This issue presents the storage administrator with the challenge of moving large amounts of data without affecting the operating hosts and guests.

The following options are available:

- ▶ One approach to address this requirement can be to always use logical volumes. With logical volumes, you can add the new disk to the server, define it as a physical volume, and add it to a volume group.

Next, you can use `pvmove` to empty an older volume and then, remove that older volume from the volume group and any system definitions. This process works for data logical volumes and the root file system logical volume.

When working with the root file system logical volume, you must take care to rebuild the `initrd`. This process often is accomplished by using a single command, such as `dracut`.

Note: Because a mistake in the process means that you render your system unable to start, be sure to test the process in a sandbox environment and be meticulous about the steps. The use of this approach means that you install your Linux systems with LVM from the beginning. Converting to LVM later is time-consuming.

- ▶ If you do not want to use root LVM, one other possible solution is a raid 1 software mirror with `mdadm`. Install Linux on to this mirror device. Then, you can add storage devices as part of the raid 1 mirror, allow them to sync and then, break the mirror and remove the older device.

However, `/boot` might not support either or both of these methods in your distribution. In this case, you must manually copy the `/boot` partition to the new disk.

Important: The key is to plan ahead and test.

2.2.3 Network considerations

From a KVM host perspective, you can choose from the following physical and virtual network devices:

- ▶ OSA-Express features (1 GbE, 10 GbE, 25 GbE, and 1000Base-T). They can be shared to guests through MacVTap or with an Open Virtual Switch (OVS).
- ▶ RoCE Express: RoCE is RDMA over Converged Ethernet. These cards can operate in three different protocol modes (TCP/IP, RDMA, or SMC-R). They are available in 10 GbE and 25 GbE options.

Note: SMC-Rv2 defines the updates to the SMC-R protocol, which allows SMC-Rv2-enabled hosts to connect and communicate over multiple IP subnets. SMC-Rv2 uses the RoCEv2 protocol that also is known as *routable RoCE*.

Open Virtual switch is *not* an option with these cards, but they can be virtualized to the guest with PCI pass-through. SMC-R operating mode includes restrictions, such as no routing and TCP is the only protocol that is supported. SMC-R can be used by a KVM host and a KVM guest. RoCE can be attached by way of MacVTap and PCI pass-through. (MacVTap offers better performance as of this writing).

From a host perspective, you might use it for live migrations or for a network shared file system. From a guest perspective, many more use cases might be available.

Databases connections, large file movement, and network shared file systems might be examples of RoCE use cases.

- ▶ HiperSockets: A virtual hardware device for high-speed, low-latency transfers with large MTU sizes possible. These HiperSockets are restricted to “in the box” communications between partitions.

Although KVM does not virtualize a HiperSockets device, they can be accessed by a KVM guest by way of MacVTap. Open Virtual Switch (OVS) also is an option for HiperSockets if the system supports VNIC characteristics.

- ▶ Internal Shared Memory (ISM): This virtual hardware device can be used directly by the KVM host. Virtualization to the KVM guest is *not* available as of this writing. Some possible use cases for a KVM host with SMC-D exist.

Live migration of guest systems or network shared file system might be ISM within a KVM host user.

The KVM hosts and guests can use connections in trunk mode with multiple VLAN IDs or an access port connection with only a single VLAN ID. Linux and KVM also can virtualize further with VXLAN support. VLAN support at most 4096 unique values or virtual LANs. VxLAN support 16 million logical networks.

The following networking constructs in the KVM host are available that can be used to connect the guest to a network:

- ▶ The default network

The KVM default network is in place with every KVM installation. It uses a nonroutable IP address range and is DHCP assigned by the KVM host.

It adds a layer of simplicity and isolation for KVM host and guest communications. Theoretically, a guest can have this one network and use NAT in the KVM host firewall.

However, users have no way to access this guest who is not on the KVM host. It also can be useful in Disaster Recovery situations where a known IP and access method is available.

- ▶ Open Virtual Switch

This production quality virtual switch supports KVM host and guest connections, tunneling between switches, trunking, bonding, link aggregation, VLANs, VxLANs, and access. It also is programmable. OVSs can have one or more OSA-Express connections, but do not require any network adapter if an isolated network is required.
- ▶ MacVTap

This construct provides the virtualization of a network device to a guest. It often features fewer layers of code than a virtual switch so that it might be better performing, but few functions are offered. MacVTap can be combined with other technologies, such as VLANs, VxLANs, or bonding.
- ▶ Bonding

The bonding construct combines multiple network adapters to provide enhanced availability, capacity, or both. Many bonding modes are available, and it also can incorporate link aggregation.
- ▶ Teaming

This construct is newer than bonding and is similar to bonding in some ways but uses a different implementation. OSA-Express or RoCE features can use bonding or teaming.
- ▶ Link Aggregation

This construct combines multiple network interfaces for enhanced availability and bandwidth. OSA-Express features that are used for link aggregation must be dedicated to a single logical partition (LPAR). For this reason, link aggregation is an expensive choice. Bonding mode 6 can provide some similar functions without dedicating OSA-Express ports to LPARs.
- ▶ Bridges

Bridges connect to different networks in a protocol-independent way. The KVM default network is a form of a bridge.

2.2.4 Encryption considerations

If you are planning to pervasively encrypt KVM hosts and guests, plan for at least one pair of Crypto Express features and a usage domain per guest. A Crypto Express feature that acts as a hardware security module (HSM) *cannot* share its domains between guests.

The master keys *must* be set for a Crypto Express adapter that is running in EP11 or CCA modes. For the EP11 mode, a trusted key entry (TKE) workstation *must* be used.

A TKE is optional for CCA mode, but recommended. The alternative is to use the `panel.exe` program from the Linux command shell to set the master keys.

Support is available for pervasive encryption of swap devices with Ephemeral keys. Because the swap data never lives across restarts, a persistent key is not needed.

ECKD and FCP attached SCSI LUNs can be used with `dm-crypt`. They both support the use of protected keys for encryption. Clear keys also are possible, but you run the risk of exposing the key.

The default mode of any Linux is to deploy with SSH, which is encrypted. Applications, such as telnet or FTP with unencrypted protocols (“in the clear”), must never be installed to avoid any accidental exposure.

VM Manager operates over an SSH connection so that communication also is encrypted.

When planning to move a large amount of data over an encrypted connection, such as a guest live migration over SSH, ensure that you use ciphers that are accelerated by CPACF. Doing so reduces CPU consumption and reduces the time that is required for the encryption operations.

For more information about encryption, see *Getting Started with Linux on Z Encryption for Data At-Rest*, [SG24-8436](#).

2.2.5 KVM guest domain considerations

Consider the following points before you start your deployment of KVM guests:

- ▶ Initial and maximum number of virtual CPUs.
- ▶ Relative share amount of CPU if you do not want the default.
- ▶ Any caps on CPU consumption.
- ▶ Initial and maximum amount of virtual memory.
- ▶ Default 4 K pages or huge pages. For maximum benefit, enable in the KVM host, KVM guest, and any using middleware, such as IBM Db2 or Java.
- ▶ Installation source: Virtual CD-ROM, network based, or cloned disk.
- ▶ Type, number, size of disks for storage: Image files, Block devices, Partitions of block devices or LVMs.
- ▶ Use of multiple I/O threads, for example:

```
<domain>
  <iothreads>2</iothreads>
  ...
</domain>
```

allocates two I/O threads for the QEMU process, and

```
<devices>
  <disk type='block' device='disk'
    <driver name='qemu' type='raw' iotthread='2' />
  ...
  </disk>
  ...
</devices>
```

- ▶ Number and size of swap devices (full device, partition, file, or LVM).
- ▶ Multipathing: No disk multipathing is required or needed when it is provided by the KVM host or IBM Z hardware configuration.
- ▶ Networking: KVM default network, MacVTap, or Open Virtual Switch, bonding, VLANs. The use of a HiperSockets LAN, SMC-R by way of RoCE, or TCP/IP by way of OSA-Express.
- ▶ vfio-ap: Crypto Express adapter AP queue.
- ▶ Virtual consoles: How many virtual serial consoles? Graphical or text based?
- ▶ CPU mode/model support. Setting the CPU mode or model support can be important to successful live migration where you have different model of s390x processors as source and target, particularly when you must migrate to an older generation of technology.

- ▶ Start menu, start order, and load parameter support, as shown in the following example:

```
<cpu mode='host-model' />
Boot menu, boot order, and load parameter support
<bootmenu enable='yes' timeout='3000' />
<disk>
...
<boot order='1' loadparm='2'>
</disk>
```

- ▶ On crash actions to take a memory dump, for example.
- ▶ Any time of day clock offset on the initial time that is used when the domain starts. Important when hosting servers in multiple time zones.
- ▶ Watchdog (Diag288) and trigger action: If you need to ensure that a guest domain is reset or dumped if the Linux inside is unresponsive.
- ▶ QEMU guest agent: This agent can provide enhanced management of a guest.

How to set the QEMU guest agent is shown in the following example:

```
<channel type='unix'>
  <target type='virtio' name='org.qemu.guest_agent.0' />
</channel>
```

Software consideration for Linux guests

In general, no special software considerations exist for a KVM guest. The Linux distribution includes all of the required drivers, such as virtio. Optional packages, such as the guest agent, are available but not required. From a separation of duties and isolation perspective, the package is not suitable where the KVM host administrator should not see into a KVM guest.

You can plan to use a FreeIPA client or an LDAP client for authentication or identity management, but that aspect is *not* unique to KVM.

2.2.6 Methods for installing Linux into a guest domain

Installing Linux as a guest can be achieved by using one of the following methods:

- ▶ You can copy Linux image (a block device or image file). Bring that copy up and tailor the hostname, IP address, and regenerate the SSH keys. The copy can be from a VM Manager “Clone” operation, an OpenStack instance request, or a lower-level Linux file `cp` command or `dd` command for a block device.
- ▶ You can install a fresh copy of Linux from scratch, which can be done by using several methods, including the following examples:
 - Graphically by using a VM Manager request for a new VM. Virt Manager prompts you if you want to install from local media (ISO image or CD-ROM), a Network Install (HTTP, HTTPS, or FTP), or Network Boot (PXE).
 - `virsh` from the command line. By using this method, an ISO image most likely is being used as a virtual CD-ROM for the installation. With `virsh`, the `--console` option gives you a text console that is used for installation.

By using this method, the domain or guest must be defined in advance, which indicates that you want to start from the virtual CD-ROM. Other installation source methods are possible.

- `virt-install` is single command-line installation method that like `virsh`, this approach defines the domain or guest and starts the installation by using one command.

Depending on your Linux distribution, the answers to the installation questions can be fully automated. Whether preseed, kickstart, or autoyast, you can describe the software packages, disk and file systems, network configuration, and remaining questions in an automated manner.

After the basic Linux operating system is installed, you can automate more tailoring by using tooling, such as Chef, Ansible, Puppet, and SaltStack. These options might tailor the configuration file in `/etc/` to meet shop standards, and perform a silent installation of middleware installation, such as IBM WebSphere Application Server or IBM Db2.

2.2.7 Linux virtual machine live migration

Planning for live migration involves the following key items:

- ▶ If different processor technology generations are used, you must configure the guest on the newest generation to operate only at the generation to which you might want to migrate them. If you fail to make this configuration, you see that the guest is migrated, but is unresponsive and shows a crashed state.
- ▶ Coding the correct processor generation is simple but must be done before you start the guest domain. Ensure that source and target KVM hosts can access the same network and storage resources and are named and addressed consistently. Naming and addressing conventions must be well defined and in place.

For example, if two machines used different device numbers for OSA-Express ports, such as device 100 on machine A and 600 on machine B, direct references to these OSA-Express ports by device number are problematic during integration.

The same is true for accessing disk storage. You always want to use a name that represents a multipathed device, such as `mpatha` or `mpathb`, and not individual paths, such as `sda` or `sdb`.

For purposes of live migration, you also want a name that includes the disk UUID. The idea is that this UUID always is consistently named across KVM hosts. Although a name, such as `mpatha`, can be persistent regarding the underlying device across restarts, `mpatha` on Hypervisor 1 can be a different device than that of Hypervisor 2.

Consider the following points:

- ▶ If you use MacVTap, you cannot live migrate to another LPAR and use the same OSA-Express port. Migration to a partition on a different machine or a different OSA-Express port on the same machine supports live migration.

Note: If the system supports VNIC characteristics for OSA, it is possible to migrate with a shared port even for MacVTap.

- ▶ If you use Crypto Express features to protect keys with a master key in an HSM, you must have Crypto Express features with the same master keys and domains on the target of the migration.

The general assumption with live migration is that the disk is shared, and you do not copy the disk because of the time and resources that are required for a copy operation. For this reason, block devices and not image files are generally considered for live migration.

However, image files can be used regarding live migration and not copying to them. You need a shared file system across the source and target KVM hosts for the image files in question. IBM Spectrum Scale (GPFS) is one such file system.

Another file system that you might use is NFS v4; however, it is often preferred to have direct Fibre Channel connectivity to a TCP/IP-based network connection for storage.

KVM live migration does not require any IBM Z channel-to-channel connections. It requires only a network connection. The fastest low latency connection that you can find is ideal. Therefore, you might consider the use of an Internal Share Memory (ISM), RoCE Express adapter, or a HiperSockets LAN for the live migration. Connections that are over OSA-Express features also work.

2.3 Planning for management and monitoring

You can manage KVM hosts and guests by using various methods. Not all environments operate in a cloud model. Your scalability demands and security requirements for KVM hosts and guests might also dictate how you manage the environment.

2.3.1 KVM host management

Common KVM host system administration tasks include the following examples:

- ▶ Stopping and starting the KVM host
- ▶ Patching the KVM host with updates
- ▶ Obtaining a console for the KVM host
- ▶ Adding or removing hardware resources from a KVM host

Stopping a host can be started within Linux by using a **shutdown** command; or, if Linux is configured to accept the signal, a STOP operation in the partition on a DPM enabled HMC shuts down Linux. By using the Linux shutdown command with the **-r** (Reboot) option eliminates the need to use the HMC to start the KVM host.

A KVM host always is started from the HMC. It also uses the Operating System Message dialog for the initial console messages.

Patching a KVM host to a newer level is handled much like any other Linux. Ideally, you take backups and shut down or relocate all running guests to other KVM hosts before you begin. Some Linux distributions incorporate file system snapshot technology to allow you to start from or roll back to a previous snapshot if the updates become problematic. The act of updating is typically accomplished by using **zypper up**, **dnf upgrade**, or **apt update** and **apt upgrade**.

Obtaining a console to the KVM host is done by using the Integrated ASCII Console from the HMC.

Adding resources to a KVM host often involves adding logical processors, memory, network interfaces, or storage groups to the partition from the HMC. To add memory dynamically, an amount must be configured that is a maximum amount greater than the initial amount.

After the extra resource is configured to the logical partition, that resource must be brought online in Linux. In the case of logical processes, CPUs must be rescanned and configured online. In the case of memory, the **chmem** command is used.

In the case of network or storage resources, new CHPIDs or devices might need to be varied online. Then, the resources must be configured to the system. This process might include defining a network interface with an IP address, or partition a disk, adding it to a volume group and LVM, or even installing a new file system on the device.

Just as they can be added, resources can be dynamically removed. Storage and network interfaces must be shut down and unconfigured before they are removed. Memory and CPU resources must be taken offline before they are unconfigured from the partition also. After Linux removes the resource, that resource can be unconfigured from the partition at the HMC.

You can also use tools, such as Cockpit, to manage a Linux guest, and that Linux can a KVM host. Cockpit features a simple to use web UI so it is accessible from any desktop platform.

2.3.2 KVM host monitoring

Several types of monitoring exist, and it is likely that more than one is implemented. You might use operational monitoring to determine whether a resource is up or down, near full or empty. You might use intrusion detection monitoring. Performance monitoring and a variation of performance monitoring are available for longer-term data for capacity planning purposes. Commercial and open source versions of these tools also are available to help you monitor the KVM host.

Tools, such as Nagios and Icinga, can perform several different checks of your Linux and KVM environment. A “check” does not exist, and tools are available that can be easily updated to include new ones.

You can opt for analytics suites, such as Elastic Search, Logstash, Kibana (ELK) or Graphite, and Prometheus and Collectd. Both stacks are well suited for visualizing time series data.

For a point-in-time look at KVM performance. `virt-top` and `kvm_stat` might be the best tools to use. Both tools look at real-time data and log data for longer term trend analysis or for graphing. Standard Linux performance tools, such as `top` and `vmstat`, are also relevant. You also can monitor key metrics by using the Cockpit web UI.

Long-term detailed performance data can be gathered from `systat/sar` for later analysis. It is ideal for system-level resource usage analysis.

You can use IBM Tivoli® Monitoring (ITM) to monitor a KVM host because it can monitor any other Linux system.

2.3.3 KVM guest management

Several tools are available to manage KVM guests. The following tools can manage the lifecycle functions and tuning, snapshots, and migration of servers:

- ▶ VM Manager
- ▶ oVirt
- ▶ OpenStack
- ▶ Cockpit
- ▶ `virsh` CLI
- ▶ QEMU guest agent

Start and shutdown operations

KVM guest can be marked to autostart when the KVM host is brought up. No specific sequencing is used with this method. If you want your database server brought up first, it might be the last. You can choose not to mark the guest domain for autostart and build a script to issue individual `virsh` start commands in a sequence with specific delay periods.

KVM guests can be shut down from the KVM host. If the sequence is of concern, you can script issuing individual `virsh` shut down commands in the wanted sequence. If necessary, a KVM domain can be destroyed, which simulates a power off.

Using a console

Most tools provide access to a guest console. You can use the `virsh console` command to obtain a console from a specific domain anytime the guest is running. Virt-Manager, Cockpit, and OpenStack all provide console access as well, including graphical consoles.

QEMU guest agent

This agent can provide enhanced management of a guest. For example, it can enhance `virsh` shutdown and allow `virsh setvcpus` to take virtual CPUs offline in the guest. It can also query guest file systems, IP address details, and more. Include the agent and enhance the manageability.

Cloning or removing a guest

Guests can be cloned by using the `virt-clone` command or from tools, such as Virt-Manager. Take care in setting a new hostname, IP configuration, and SSH keys. Your Linux distributor might have specific documented advice about reenabling “first start” processes.

Removing a guest is as simple as using the `virsh undefine` command or clicking any of the graphical tools.

Guest snapshots

Guest snapshots can be used to have a point-in-time copy of a guest. They can be reverted to their original state after some testing, or they can be used for backup purposes.

The following types of snapshots are available:

- ▶ Internal

These snapshots keep the snapshot inside the original `qcow2`. Internal snapshots are used by Virt-Manager.

- ▶ External

These snapshots are newer. They create a separate file to hold updates that occur *after* the snapshot. Although not managed by Virt-Manager, these CLI-based snapshots are newer than the internal version and are often preferred over their predecessor.

Note: The QEMU guest agent code is used to perform a file system freeze.

Guest save/restore

A running guest can save its state (memory, but not disk) and later be restored to resume execution. After a guest is saved, it no longer is running, which frees the memory on the system.

Guest suspend/resume

You can suspend a running KVM guest by using the `virsh` CLI or other tools, such as Virt-Manager. Although this process does not free memory, it does stop the scheduling of the guest domain until it is resumed.

2.3.4 KVM guest monitoring

Many of the guest management tools include facilities to monitor the guest in some capacity. All of the tools you use to monitor a KVM host are relevant to a KVM guest, including the following examples:

- ▶ VM Manager
- ▶ oVirt
- ▶ OpenStack
- ▶ Cockpit
- ▶ virsh CLI
- ▶ QEMU guest agent
- ▶ ELK stack
- ▶ Grafana
- ▶ IBM Tivoli Monitoring (ITM)
- ▶ Perf
- ▶ Sysstat/SAR
- ▶ top, vmstat, virt-top, and kvm_stat

2.4 Planning for security

Security too often can be an afterthought on a deployment project. More security often is associated with added complexity. However, this presumption is not always accurate.

Failing to implement centralized identity management or authentication solution can add complexity. Imagine dozens or hundreds of virtual servers, each that uses only their own `/etc/passwd` for authentication.

Also, your security policy requires frequent password changes. Whether your own authentication credentials, that of a DBA team, or an application support team, the situation quickly becomes unmanageable, not only in terms of password management, but also suitable handling of user IDs with staff turnover.

Centralized Identity management helps to better secure and to simplify this situation.

2.4.1 Access controls

Access control in Linux can be accomplished in several ways. The most common aspect that is thought of is file system access controls of Read, Write, Execute for Owner, Group, and everyone else, which also can be augmented by Access Control Lists (ACLs) and `setfacl/getfacl`.

The next layer of access control is Security Enhanced (SE) Linux and AppArmor. You implement one or another, and it places more controls on users, groups, programs, and files. Policy Kit (polkit) also can help you implement more granular controls.

PAM is another access control point that you can tailor to restrict access to Linux capabilities. Often, individual programs or applications, such as SSH, feature their own configuration files in which you can tailor more access control, such as require public key or MFA controls instead of password usage, disabling root SSH, or only allow SSH from a specific IP address or range.

2.4.2 Authentication solutions

The most basic authentication is handled by using `/etc/passwd`. However, `/etc/passwd` is not the best choice when multiple servers are used and UIDs and GIDs must be coordinated for shared file systems. Solutions, such as NFS and IBM Spectrum Scale (GPFS), require this UID and GID consistency across servers.

The most basic component of centralized authentication for Linux is LDAP. LDAP supports defining your users, groups, and their credentials, and stores sudo configurations.

One of the challenges is administering LDAP. Working with shell scripts and LDIF files can be more than some administrators can take on. FreeIPA can help to simplify this burden. FreeIPA is a solution that uses well-known open source projects, such as 389 Directory Server, MIT Kerberos, NTP, DNS, and Dogtag.

FreeIPA provides a scriptable command-line interface and a web UI. Both are key to the simplification of administering LDAP. FreeIPA also controls user access to individual hosts and services on those hosts.

FreeIPA also provides an elegant multi-factor authentication (MFA) solution, centralized sudo administration, a centralized public key repository for key-based authentication, and centralized user administration. With a mouse click, you can create password policies and apply them to groups of users.

FreeIPA includes a client and a server component. The client installation script downloads and installs the server's public certificate and configures Kerberos, `sssd`, `pam`, and `nsswitch` to use the FreeIPA server. Even if you use a Linux distribution that does not have a FreeIPA client, the client components can be configured to use the FreeIPA server.

You also can choose from commercial solutions for authentication and authorization. Solutions, such as Centrify, provide a commercially supported way to integrate with non-Linux authentication repositories.

2.4.3 Audit

Linux audit provides the means to audit almost every aspect of Linux. You can configure it in a manner that complies with your installation's security policy. Sample security policies often are provided with the Linux audit package.

You might also want to consider the amount, size, and duration of Linux audit data to retain. You can also send your audit data to a centralized audit repository in real time. Commercial products, such as IBM QRadar, are available to analyze audit and log data.

At a minimum, you expect to gather and retain system logs for audit purposes. Although they might not contain everything that you need, they do include a significant amount of information.

The `rsyslogd` can be configured to transmit log data in real time to a centralized log repository and send them over a TLS encrypted connection.

If you use other products or commercial middleware, they also must create logs that are retained for audit purposes.

2.4.4 Firewalls

Firewalls play a key role in Network security. Combining a firewall with Host-Based Intrusion Detection tools allows you to respond quickly and precisely to a network-based threat.

Fail2ban is an example of an application that scans your logs to look for abnormal conditions and blocks the offending IP from the application it is probing. The old way of thinking of “If a firewall is in the DMZ, I am protected,” is not good enough.

Because today’s network communications are encrypted, those external firewalls are unaware of the requests are being made to the application on your virtual server. However, the application logs do indicate security events, such as invalid user ID or password. They also can show the user ID that is being attempted and that a brute force attack is in progress.

Several other intrusion detection tools also are available in addition to fail2ban, including OSSEC, AIDE Snort, Sagan, Suricata, and Samhain. Consider the use of these open-source network and host-based intrusion detection tools.

2.4.5 Cryptography

Cryptography adds the data privacy and protection aspect to security. It generally encompasses network traffic (data in-flight) and storage (data at-rest). IBM Z cryptographic hardware can reduce CPU consumption, accelerate cryptographic operations, and provide a tamper resistant master keystore.

IBM Z hardware offers two key security components: the CPACF and the Crypto Express adapter. Virtualization of the CPACF takes no special planning beyond ensuring that the enabling microcode feature is in place. A Crypto Express adapter features multiple domains that can be assigned by the KVM host to a guest. These Crypto Express adapters are referred to as *Adjunct Processor (AP) queues*.

KVM guests access AP queues through an AP Virtual Function I/O (VFIO) mediated device. Configuring the mediated device defines the AP configuration of the KVM guest to which it is assigned.

For more information, see *Configuring Crypto Express Adapters for KVM Guests*, [SC34-7717](#).

When operating as an HSM in CCA or EP11, the Crypto Express adapter domains or AP queues *cannot* be shared between virtual servers. This point is key to planning. If you have more than 85 Linux instances that need crypto domains, you need another pair of Crypto Express adapters. A minimum number of two adapters are recommended for redundancy if a feature must be serviced so that an outage can be avoided.

The Crypto Express adapter must have each domain initialized with unique master keys. This process can be done in Linux by using a command-line program or by using a Trusted Key Entry (TKE) workstation. A TKE is required for EP11 mode, and optional for CCA mode. The use of a TKE is recommended for production environments.

You also must plan to include software libraries on your Linux systems to use the cryptographic hardware. OpenCryptoki and libica are two of the common libraries. The selection of the cipher that is used by software (such as OpenSSH or OpenSSL) can affect whether the CPACF or Crypto Express adapter is used. Many, but not all, Cipher or Hash are implemented in the hardware.

You can use this safeguard for data that you store on FICON ECKD and FCP SCSI disk storage. Also, plan to use this protection for your swap devices if security is of any concern.

For more information about implementing pervasive encryption with Linux on IBM Z, see the following publications:

- ▶ *Pervasive Encryption for Data Volumes*, [SG24-2782](#)
- ▶ *Getting Started with Linux on Z Encryption for Data At-Rest*, [SG24-8436](#)

2.4.6 Multi-factor authentication

Enabling technology for MFA was around for years and included published RFC standards HOTP and TOTP. Evolving security requirements and regulations are rapidly driving MFA to a standard authentication model for any professional and credible IT organization.

Many open-source solutions are available in this space. These solutions include Basic MFA deployments that use Google Authenticator and `google-authenticator-libpam` or more robust solutions that incorporate MFA into the overall identity management solution, such as FreeIPA.

You also can choose from commercial solutions, such as Centrify. Commercial solutions might provide their own modules, perhaps for the PAM layer, or plug into other standard access points in Linux, such as with Radius.

2.5 Planning for backup and recovery

It also is important to periodically test the backup and restore processes. Each layer (hypervisor, Linux guest, middleware, and applications) might require its own backup/restore processes, and they are likely different.

At the hypervisor and guest layers, you often find file-level backups or full block device backups. The use of point-in-time copy technology, such as FlashCopy or KVM snapshots, is critical to having a viable backup.

The alternative is to shut down the systems to obtain the required consistency to be sure that a viable backup exists. File-level backups are useful for restoring a single or group of files, but not an entire block device. They are not helpful when your server does not start.

At the middleware layer, you might have transaction logs and database backups, which are preferred for things such as databases. Although less frequently required, an application might need its own backup process. This need might be more true for purchased applications that implement their own backup/recovery process. These backups must be held on an independent storage device and ideally, retained at a different physical location.

Keeping backups inside the same storage server cannot be considered safe or acceptable practice.

2.5.1 KVM host backups and recovery

The KVM host backup can be broken down into the following main categories:

- ▶ The core operating system disk that is needed to start
- ▶ The extra storage that is used to host image files and system logs
- ▶ Key configuration files, such as for networking and virtual server definitions

At a minimum, the KVM host includes point-in-time copy backups of its block devices so it can start if it is damaged. File-level backups of configuration files and logs are helpful, but cannot always be used to resurrect a KVM host that cannot be started.

Each of these components can be backed up by using several methods. The core operating system disk in its most basic form can be backed up by running the Linux **dd** command from another system. You might want to run this command immediately after installation.

You also can use FlashCopy or disk mirroring technologies to create a consistent point-in-time copy without taking down the KVM host. To use FlashCopy or similar technology, it might be required that a CLI program (for the IBM DS8000® family) is installed to direct the FlashCopy operation, and to enable network connectivity to the console of the storage subsystem.

The extra storage that is used to host image files also can use FlashCopy or disk mirroring, but other options also exist. A QCOW2 snapshot or an LVM snapshot are examples of other options that might help you minimize downtime.

Key configuration files, such as the KVM host network definitions, OVS definitions, `zipl.conf`, `zfcplib.conf`, and others, can be backed up through file-based tools, such as `rsync` or commercial products (for example, IBM Spectrum Protect). The amount of storage these configuration files use is relatively small.

In addition to backing up the block devices that the hypervisor requires, regularly gather metadata that is helpful to a recovery process. This data can be easily gathered by using a shell script.

The **df -h** command can provide needed file system details. The **pvs**, **vgs**, and **lvs** commands can provide needed information about Logical Volume configuration. The **multipath -l** command can show disk UUID and naming information.

Understanding the contents of `zipl` and the last IPL command-line arguments also are important. Other metadata that is related to TCP/IP configuration also might be helpful. Gather in advance all of the data that you might ever want to perform a recovery.

Concerning the point-in-time copy, operations “consistency” across the multiple disks that the server is composed of is important. A copy of each disk from an LVM at slightly different times can lead to corruption on a restored system. It is important to place all of the disks into a consistency group so that the copy of all of the disks is from the same time. Then, back up those copies to another location.

FlashCopy commands can be issued by using a shell script and **cron**. Plan to have another server dump the block devices by using the **dd** command. They can be `gzipped` and transmitted to another location by using SSH over the network.

Naming and logging the backups can be important. Keeping multiple generations of those backups also is necessary. With only a single backup, you risk having a backup that can be corrupted from which you cannot recover.

Restore simplicity also is important. Restoring an individual file or group of files often is not an issue. Restoring full block devices also can be simple. However, if you partition a large block device and give different partitions to multiple different guests, this process adds complexity to a restore of that one large block device.

You might find it helpful to use a virtual server that is dedicated to handling backups if you want to transmit copies of the FlashCopy targets to another location over the network. You also might want to configure this server to ignore the volume groups and LVMs names from the other servers.

Finally, take the time to test recover the system from backups. It is important to perform this test periodically. Environments, the configuration, and even the programs that are used to back up and recover change over time. Testing the recovery validates the tools and processes from end-to-end.

If the recovery is not tested, you cannot depend on it to work when you need it.

2.5.2 KVM guest backup and recovery

The KVM guest can be backed up in different ways. These methods can allow the guest to be running or require it to be shut down for data integrity purposes.

The KVM guest does include similarities to the KVM host in how you might backup and recover it. One key difference is the KVM guest might use image files. If it uses only image files and no block devices, KVM Snapshots can help to simplify taking point-in-time copies of the virtual server.

If the server uses all or some block devices, having a point-in-time copy of them also is important. For IBM DS8000 family of storage, this point-in-time copy uses FlashCopy. IBM FlashCopy works with ECKD and SCSI LUNs. For an IBM DS8000 family device, use the DSCLI; for IBM Storwize® family devices, use a shell script with SSH.

For file-level backups, you can use open-source tools, such as rsync or commercial tools (for example, IBM Spectrum Protect). If a KVM guest is destroyed, one approach might be to reprovision the guest from a Linux image and restore all the files from the most recent backup rather than the use of disk image-level backups and restores.

From a metadata perspective, you might need to have a copy of the domain XML for recovery. As with the KVM host, the other details, such as the file systems, LVMs, disk UUID, boot loader configuration, and network details, also must be collected.

Part of the planning for backup and recovery also must consider the middleware. For example, a database typically uses its own utilities to provide backups without any or minimal downtime.

A comprehensive backup and recovery strategy often involves multiple backup methods and the recovery from those backups must be regularly tested.



Preparing the Red Hat Enterprise Linux Kernel-based Virtual Machine environment for virtual machine use

This chapter provides instructions to perform an installation of Red Hat Enterprise Linux (RHEL) on a logical partition (LPAR), prepare it as a Kernel-based Virtual Machine (KVM) host, and deploy KVM guests.

This chapter includes the following topics:

- ▶ 3.1, “Defining the target configuration” on page 48
- ▶ 3.2, “Preparing the infrastructure” on page 50
- ▶ 3.3, “Collecting information” on page 53
- ▶ 3.4, “Installing RHEL on an LPAR as KVM host” on page 57
- ▶ 3.5, “Configuring the KVM host” on page 61
- ▶ 3.6, “Deploying virtual machines on KVM” on page 80

3.1 Defining the target configuration

To prepare the environment for the workloads that run in the virtual machines (VMs), it is recommended to build an installation plan. For more information, see Chapter 2, “Planning for the Kernel-based Virtual Machine host and guest” on page 21.

This chapter provides the instructions to configure and deploy a basic KVM environment on RHEL.

3.1.1 Logical View

The Logical View of our lab environment that is used in this book is shown in Figure 3-1. This view provides an overall view of the entire environment and can be built during the planning phase. For more information, see Chapter 2, “Planning for the Kernel-based Virtual Machine host and guest” on page 21.

The following networks are available for guests, as described in 3.5.8, “Defining the MacVTap network” on page 74:

- ▶ External network through two MacVTap networks
- ▶ Internal Z platform network through the HiperSocket MacVTap network

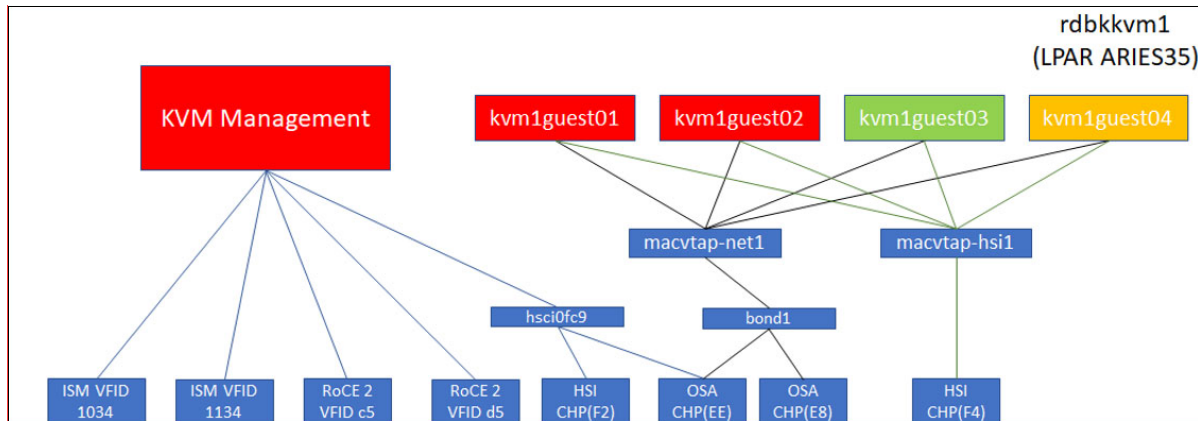


Figure 3-1 RHEL logical view

The KVM hosts access the following networks:

- ▶ HiperSockets network through an HSI0 interface.
- ▶ Internal Shared Memory (ISM) or (SMC-D), as described in Chapter 2, “Planning for the Kernel-based Virtual Machine host and guest” on page 21.
- ▶ RoCE network (SMC-R), as described in Chapter 2, “Planning for the Kernel-based Virtual Machine host and guest” on page 21.
- ▶ External network through an Open Systems Adapter (OSA) network interface card (NIC).

3.1.2 Physical resources

In this section, we describe the hardware and connectivity setup, as shown in Figure 3-2.

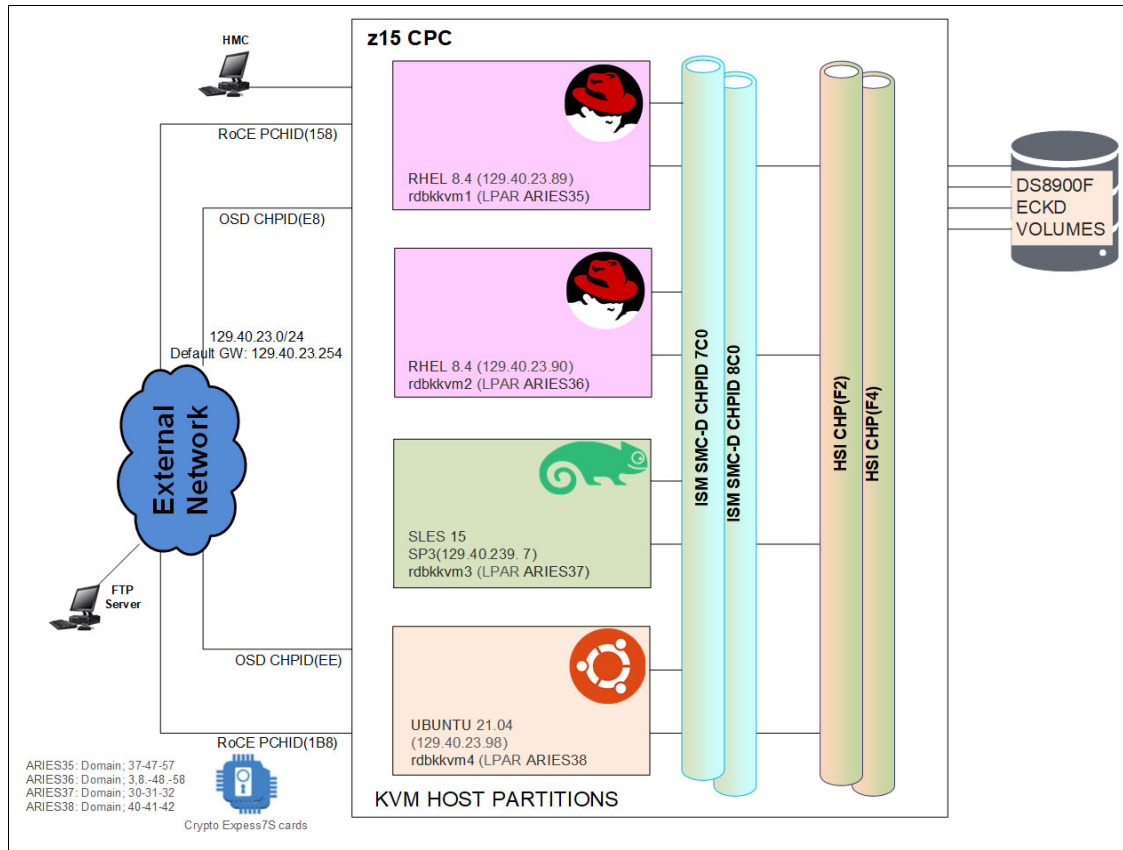


Figure 3-2 RHEL LPAR resources

The hardware and configuration setup consists of the following components:

- ▶ One IBM z15 platform with four logical partitions (LPARs)
- ▶ Two OSA cards that are connected to a LAN network
- ▶ Two Fibre Connection (FICON) cards for connectivity to storage: Small Computer System Interface (SCSI) devices (FICON as FCP adapter)
- ▶ Four FICON Express16SA+ for connection to the ECKD DASD on IBM DS8900F storage box
- ▶ One FTP server
- ▶ One HiperSocket CHIPD
- ▶ One ISM defined as SMC-D
- ▶ Two RoCE cards that are defined as SMC-R
- ▶ Four Crypto Express cards

All LPARs can access all resources. This lab includes the following LPARs:

- ▶ ARIES35: For RHEL
- ▶ ARIES36: For RHEL
- ▶ ARIES37: For SLES
- ▶ ARIES38: For Ubuntu

This chapter focuses on the RHEL implementation.

3.1.3 Software resources

For our configuration, we chose RHEL 8.4.

Note: The operating system architecture of the Z platform is s390x and the Linux packages must be based on this architecture.

For more information about RHEL supported versions on IBM Z and Z platform, see this [IBM Documentation web page](#).

For KVM virtualization (beyond the operating system, the virtualization package is required for a KVM host. For more information, see the [Red Hat documentation](#).

3.2 Preparing the infrastructure

The IT infrastructure planning depends on many factors, as discussed in Chapter 2, “Planning for the Kernel-based Virtual Machine host and guest” on page 21.

During the planning phase, we made some decisions regarding the IT resources that are needed for our lab environment. The following sections are based on those decisions.

3.2.1 Configuring the resources in Z platform

For this book, we used the traditional tool, Hardware Management Console (HMC) and Input/Output Configuration Data Set (IOCDS) to set up the resources.

For more information about IOCDS, see *I/O Configuration Using z/OS HCD and HCM*, [SG24-7804](#).

3.2.2 Configure the storage resources

In our lab configuration (see Table 3-1), we used the ECKD DASD configuration as storage devices for the KVM and the guest storages. You also can use SCSI LUNs by using Fibre Channel Protocol (FCP) configuration, as described in 2.2.2, “Storage considerations” on page 27. On IBM Z, the ECKD disk is accessed through its device address. After we formatted the device under Linux, the disk shows a name that starts with a `dasdx` prefix where `x` can vary from `a` - `z`.

Table 3-1 Lab environment ECKD DASD details

Device address	Volume name	Capacity	Description
90DD	dasda	400 GB	rdbkvm1 start and root disk
91A8	dasdb	54 GB	Volume group for KVM guest qcow2 files
91A9	dasdc	54 GB	VFIO DASD for kvm1guest3

If you use the FCP SCSI LUN environment, you must work with the storage team to prepare the disks. The worldwide port name (WWPN) must be provided to the storage team for the correct SAN zoning configuration. An example of WWPN information that is needed for the zoning is WWPN of the IBM Z FCP channels and the storage target ports, as shown in Example 3-1.

Example 3-1 SCSI storage example

-
- o FCP subchannels WWPN:
C05076D08001D9A0 is the WWPN for B908 device.
C05076D080009220 is the WWPN for C908 device.
 - o Storage target PORTS:
5005076309141145 is the WWPN for P1 storage device port.
5005076309149145 is the WWPN for P2 storage device port.
50050763091b1145 is the WWPN for P3 storage device port.
50050763091b9145 is the WWPN for P4 storage device port.
-

Figure 3-3 shows an FCP/SCSI storage area network (SAN) configuration example.

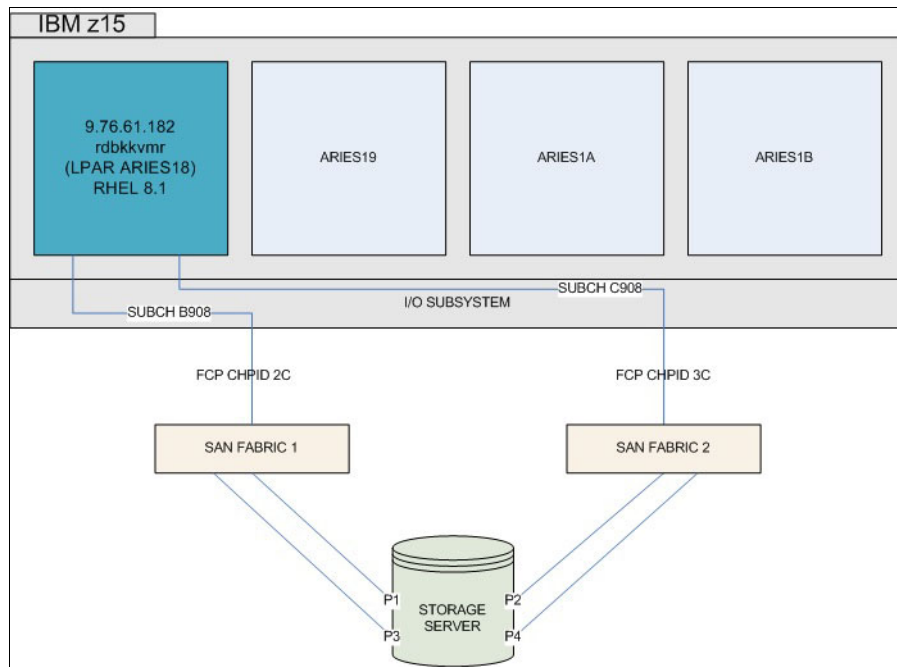


Figure 3-3 RHEL SAN configuration example

Setting up the FTP server for the installation

In this example, by following the Red Hat that are instructions that are found in the Red Hat documentation, [Installing in an LPAR](#), we create a directory in our FTP server with an IP address of 9.76.56.32, download the .ISO from the RHEL portal and then, upload the content to the FTP server.

Under the directory RHEL-8.4, after the .ISO is available and the FTP server is accessible by the target (HMC or DPM consoles), the host operating system can be installed by choosing the FTP method of installation.

FTP can provide a secondary function, which is a local packages repository. The following files in the RHEL-8.4/ directory structure are required for the packages installation:

- ▶ /AppStream
- ▶ /BaseOS
- ▶ images/
 - TRANS.TBL
 - cdboot.img
 - cdboot.prm
 - generic.prm
 - genericdvd.prm
 - initrd.addrsize
 - initrd.img
 - install.img
 - kernel.img
 - rdbkkvm1.prm
 - redhat.exec
- ▶ boot.catalog
- ▶ extra_files.json
- ▶ generic.ins
- ▶ media.repo
- ▶ rdbkkvm1.ins
- ▶ EULA
- ▶ GPL
- ▶ RPM-GPG-KEY-redhat-beta
- ▶ RPM-GPG-KEY-redhat-release
- ▶ TRANS.TBL

3.3 Collecting information

Based on the instructions that are provided in the planning stage as described in Chapter 2, “Planning for the Kernel-based Virtual Machine host and guest” on page 21, it is recommended that you save the information you use during the installation process.

A good practice is to create a checklist table (see Table 3-2) that includes the component’s information. This information is useful during the installation process and for future consultations.

Table 3-2 Sample KVM host installation checklist

Name	Type	Description	More information
Host IP/Subnet	TCP/IP	129.40.23.90	KVM Host
	VLAN	8	
Hostname domain	DNS	rdbkkvm1.pbm.ihost.com	DNS Server 120.40.106.1
Gateway	Default GW	129.40.23.254	
FTP Server	FTP port 20/21	rdbkftp1.pbm.ihost.com	Check firewall rules
FTP folder	install folder	/RHEL-8.4	Check Permission
FTP Access	Credentials	User: Inxadmin pw xxx	
LPAR	Logical Partition	ARIES35	
Memory	RAIM Memory	256 GB	HostOS, GuestsOS, and Workloads
Physical Processors	IFL (shared)	16 IFLs	SMT enabled
Virtual Processors	Virtual Processors	Two for each guest	Can be expanded later, recommended vCPU number <=max of physical CPUs
Storage	ECKD DASD	0.0.90DD 0.0.91A8 0.0.91A9	400 Gb 54 Gb 54 Gb
SCSI WWP1	FCP HBA B908	C05076D08001D9A0	PORT1: 5005076309141145 PORT2 : 5005076309149145
SCSI WWP2	C908	C05076D080009220	PORT3: 50050763091b1145 PORT4: 50050763091b9145
OSA1	Network card 1	CHP E8	Devices 1E80-1E82
Crypto	Domain / Card	CARDS 0x03 0x06	DOMAINS 0x25 0x2F 0x39N

3.3.1 Installing RHEL on an LPAR installation

In this section, we provide information about our lab environment. You can use this information as a reference to create your own environment.

Installing by using FTP

RHEL can be installed from a DVD in the HMC or from an FTP server. In this example, we decided to install RHEL from an FTP server. Be sure to have the FTP port open in the firewall. The following FTP server information was needed in our lab environment:

- ▶ IP address: rdbkftp1.pbm.ihost.com
- ▶ Credentials: User lnxadmin and password ftppass
- ▶ Directory: /RHEL-8.4

OSA device addresses

On the IBM Z platform, the network interface cards (NIC) are represented by OSA express adapters. Each OSA card can manage a range of devices. The use of a specific OSA requires three consecutive addresses: one device for control reads, one for control writes, and another for data.

For this example, we chose the first triplet from OSA CHPID E8 (1E80-1E82).

Networking information

Contact your network administrator to obtain the correct networking information for the host.

Our lab environment included the following networking information:

- ▶ Hostname: rdbkkvm1
- ▶ IP address: 129.40.23.89
- ▶ Subnet prefix: 24
- ▶ Default gateway: 129.40.23.254
- ▶ Layer 2 or 3: 2
- ▶ VLAN: 8
- ▶ DNS: 129.40.106.1 and 129.40.106.2

Note: IP address 100.150.233.40 was used for HiperSockets network access.

Storage

As described in 2.2.2, “Storage considerations” on page 27, two options are available for storage on the Linux on IBM Z platform: ECKD DASD disk or FCP LUN disk. In this example, we used ECKD DASD.

Our example features the following storage information:

- ▶ ECKD device address: 90dd
- ▶ Volume serial: 0X90DD
- ▶ Space: 400

The operating system installation uses a single DASD under Logical Volume Manager (LVM).

3.3.2 Virtual machine installation information

In this section, we review the following required information for VM installations:

- ▶ Compute
- ▶ Memory
- ▶ Disk
- ▶ Network
- ▶ Cryptography

Compute

For VM deployment, all the guests use two virtual CPUs (vCPU) to take advantage of the Simultaneous Multi-Threading (SMT) on an IBM Integrated Facility for Linux (IFL) processor.

Memory

The amount of memory is related to the type of workload that a machine is going to host; each VM has 2 GB of RAM.

For the Linux guest operating system, we recommend starting with 512 MB of memory (see Chapter 2, “Planning for the Kernel-based Virtual Machine host and guest” on page 21).

To avoid memory constraints, it is good practice to perform an accurate workload and capacity study to correctly define the amount of memory that you need.

Disk

QEMU Copy On Write (QCOW2) is a file format for disk image files that are used by Quick Emulator (QEMU), which is a hosted VM monitor. QCOW2 uses a disk storage optimization strategy that delays allocation of storage until it is needed.

Files that are in QCOW2 format can include disk images that are associated with specific guest operating systems. QCOW2 supports multiple VM snapshots through a flexible model for storing snapshots.

A QCOW2 image file was used for the operating system disk in our example.

The files were stored in the LVM to create more flexible storage migrations. For more information, see 2.2.2, “Storage considerations” on page 27.

The ECKD DASD used for the Volume Group (VG) that is used for images (rdbkkvm1-images), is the 0X91A8 volume.

A maximum of 10 GB of space was specified in our lab environment for the image files, but can be extended. We created the following four disk images to use as storage for the VM guests:

- ▶ kvm1guest01: /var/lib/libvirt/images/kvm1guest1_vol001.img
- ▶ kvm1guest02: /var/lib/libvirt/images/kvm1guest2_vol001.img
- ▶ kvm1guest03: /var/lib/libvirt/images/kvm1guest3_vol001.img
- ▶ kvm1guest04: /var/lib/libvirt/images/kvm1guest4_vol001.img

Network

As described in “OSA device addresses” on page 54, contact your network support team to obtain the proper networking information.

The external network access setup that was used in our lab environment included the following parameters:

- ▶ Hostname: kvm1guest01
- ▶ IP address: 129.40.23.200
- ▶ Subnet prefix: 24
- ▶ Default gateway: 129.40.23.254
- ▶ Hostname: kvm1guest02
- ▶ IP address: 129.40.23.201
- ▶ Subnet prefix: 24
- ▶ Default gateway: 129.40.23.254
- ▶ Hostname: kvm1guest03
- ▶ IP address: 129.40.23.202
- ▶ Subnet prefix: 24
- ▶ Default gateway: 129.40.23.254
- ▶ Hostname: kvm1guest04
- ▶ IP address: 129.40.23.203
- ▶ Subnet prefix: 24
- ▶ Default gateway: 129.40.23.254
- ▶ For HiperSockets access:
 - Hostname: rdbkkvm1
 - IP address: 100.150.233.40
 - Hostname: kvm1guest01
 - IP address: 100.150.233.21
 - Hostname: kvm1guest02
 - IP address: 100.150.233.22
 - Hostname: kvm1guest03
 - IP address: 100.150.233.23
 - Hostname: kvm1guest04
 - IP address: 100.150.233.24

Cryptography

In our lab environment, we assigned four crypto adapters and three domains to the ARIES35 LPAR. For more information about the z15 Crypto Express adapters, see section 2.4.5, “Cryptography” on page 42.

The Adjunct Processor (AP) queues that we used in our lab environment as our virtual cryptographic resources are listed in Table 3-3.

Table 3-3 AP queues assignment

Crypto domains/ Crypto adapters	03 (0x03)	06 (0x6)
37 (0x25)	03.0025	06.0025
47 (0x2F)	03.002F	06.002F
57 (0x39)	03.0039	06.0039

As described 2.4.5, “Cryptography” on page 42, the AP queues are a combination of <crypto card>.<crypto domain> (both expressed in hexadecimal form).

Consider the following points:

- ▶ Domain 24 was used for KVM host
- ▶ Domain 73 was used for KVMRVM01
- ▶ Domain 74 was used for KVMRVM02

3.4 Installing RHEL on an LPAR as KVM host

In this section, we describe how to complete the following tasks:

- ▶ Prepare for the installation
- ▶ Install RHEL on an LPAR
- ▶ Prepare the host for virtualization

3.4.1 Preparing the installation

For more information about using an FTP server to install RHEL on an LPAR, see “Installing by using FTP” on page 54

In this example, we created a directory structure containing the `.ins` and `.prm` files that are needed for the installer for RHEL on an LPAR.

Example 3-2 shows the contents of the `rdbkkvmr.ins` file, which is a copy of the generic `.ins` file that is provided in the root of the RHEL ISO installer. Only change the line `images/generic.prm`, replacing `generic.prm` with `rdbkkvmr.prm`.

Example 3-2 `rdbkkvmr.ins`

```
* minimal lpar ins file
images/kernel.img 0x00000000
images/initrd.img 0x02000000
images/rdbkkvmr.prm 0x00010480
images/initrd.addrsize 0x00010408
```

Example 3-3 shows the contents of the `rdbkkvm1.prm` file. It defines DASD for the target installation (or the SCSI information if you have FCP SAN disk), network properties, and the location of the FTP repository.

Example 3-3 rdbkkvm1.prm for ECKD DASD¹

```
ro ramdisk_size=40000 rd.dasd= 0.0.90dd
rd.znet=qeth,0.0.1e80,0.0.1e81,0.0.1e82,layer2=1,portno=0,portname=DUMMY
ip=129.40.23.89::129.40.23.254:255.255.255.0:rdbkkvm1:enc1e80.008:none
inst.repo=http://lrxadmin:lnx4rdbk@129.40.23.88/rhe184
inst.vnc inst.vncpasswd=lnx4rdbk vlan=enc1e80.008:enc1e80
```

Statements in Example 3-4 define two different paths to the LUN.

Example 3-4 rdbkkvm1.prm for SCSI LUNS example

```
ro ramdisk_size=30000000 zfcf.allow_lun_scan=0
rd.zfcf=0.0.b908,0x5005076309141145,0x4000400500000000
rd.zfcf=0.0.c908,0x5005076309149145,0x4000400500000000
rd.znet=qeth,0.0.1e80,0.0.1e81,0.0.1e82,layer2=1,portno=0,portname=DUMMY
ip=129.40.23.89::129.40.23.254:255.255.255.0:rdbkkvm1:enc1e80.008:none
inst.repo=http://lrxadmin:lnx4rdbk@129.40.23.88/rhe184
inst.vnc inst.vncpasswd=lnx4rdbk vlan=enc1e80.008:enc1e80
```

Consider the following points if you have SCSI implementation:

- ▶ Each `rd.zfcf` statement contains three parameters, which together define a path to a LUN:
 - The first parameter defines the FCP device on the IBM Z side.
 - The second parameter defines the target worldwide port name (WWPN), which is a WWPN of disk storage.
 - The third parameter provides a LUN number, which means that the `rd.zfcf` statements that are shown in Example 3-4 define two different paths to the LUN.
- ▶ The `rd.znet` statement defines which device triplet is used as the NIC for an installer.
- ▶ The `ip` statement defines the IP properties for the NIC or the VLAN interface if you use trunk switchport.
- ▶ The `inst.vnc` and `inst.vncpasswd` statements indicate that the installation process is done through VNC.

For more information, see this Red Hat [Customer Portal web page](#).

¹ The `rd.dasd` statement points to our storage ECKD disk device.

3.4.2 Installing RHEL on an LPAR

After all of the prerequisites were met, we started from FTP by using the information as described in “Installing by using FTP” on page 54 (see Figure 3-4).

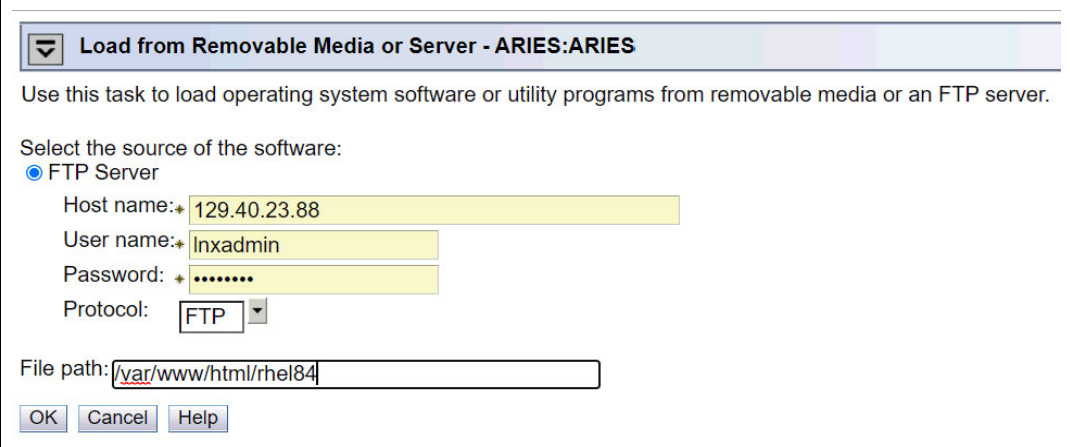


Figure 3-4 Loading from an FTP server

In the DPM or HMC, when you receive the prompt with the list of .ins files, choose the file that you created, such as rdbkkvmr.ins.

Continue with the installation process and use the [Red Hat Customer Portal](#) as guidance.

3.4.3 Preparing the host for virtualization

Complete the following steps to enable RHEL on IBM Z as a KVM host:

1. Subscribe the server to the RHEL network.

To access the packages and support, it is recommended to subscribe your system to the Red Hat Network. For more information, see this [Red Hat web page](#).

You also can install packages from a local repository. Create a file for each local repository under a repository directory, which usually is /etc/yum.repos.d (see Example 3-5).

Example 3-5 Local repository file

```
[root@rdbkkvmr yum.repos.d]# cat rhel8-dvd.repo
[rhel8-dvd]
name=Red Hat Enterprise Linux $releasever - $basearch - DVD
baseurl=ftp://itso:itso1cpo@9.76.56.32/RHEL-8.4/BaseOS/
enabled=1
gpgcheck=0
```

2. Check whether the LPAR supports virtualization functions, as shown in Example 3-6.

The LPAR must support Start Interpretive Execution (SIE) instructions.

Example 3-6 Checking virtualization support

```
[root@rdbkkvmr ~]# lscpu | grep sie
Flags:                esan3 zarch stfle msa ldisp eimm dfp edat etf3eh highgrs te
vx vxd vxe gs vxe2 vxp sort dflt sie
```

3. Load the KVM module and verify that it is loaded by following Example 3-7, which shows issuing the Linux command to load the KVM module and then, validate that the KVM module is loaded by using the `lsmod` command.

Example 3-7 Loading KVM module

```
[root@rdbkvmr ~]# modprobe kvm
[root@rdbkvmr ~]# lsmod | grep kvm
kvm                376832  20 vfio_ap
```

4. Install virtualization packages and modules.

It is important to install the virtualization modules during the LPAR installation, as shown in Figure 3-5, by selecting the `virtualization hypervisor` option in the RHEL installation process. Alternatively, you can install it later by running the `yum module install virt` command.

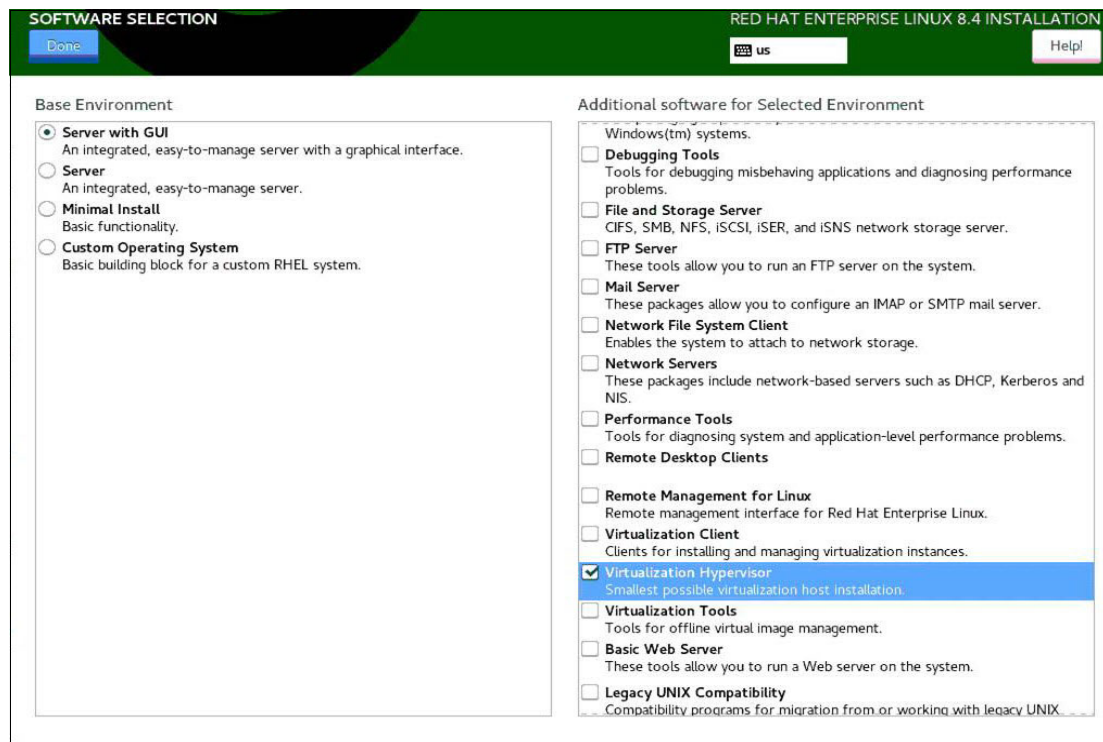


Figure 3-5 Virtualization Hypervisor option

5. After Linux is running, install the `virt-install` package, as described in 3.1, “Defining the target configuration” on page 48. This package provides new VMs by using the command line (see Example 3-8).

Example 3-8 Installing virt-install package

```
[root@rdbkvmr ~]# yum install virt-install
```

6. Validate that the host is ready for virtualization.

Before starting to work with KVM, run the `virt-host-validate` command, as shown in Example 3-9.

Example 3-9 Virtualization verification

```
[root@rdbkvmr ~]# virt-host-validate
QEMU: Checking for hardware virtualization           : PASS
QEMU: Checking if device /dev/kvm exists             : PASS
QEMU: Checking if device /dev/kvm is accessible      : PASS
QEMU: Checking if device /dev/vhost-net exists       : PASS
QEMU: Checking if device /dev/net/tun exists        : PASS
QEMU: Checking for cgroup 'cpu' controller support  : PASS
QEMU: Checking for cgroup 'cpuacct' controller support : PASS
QEMU: Checking for cgroup 'cpuset' controller support : PASS
QEMU: Checking for cgroup 'memory' controller support : PASS
QEMU: Checking for cgroup 'devices' controller support : PASS
QEMU: Checking for cgroup 'blkio' controller support : PASS
WARN (Unknown if this platform has IOMMU support)
```

You can ignore the “WARN” message that is highlighted at the bottom of Example 3-9. This message is expected and normal when installing on the Z platform.

The I/O memory management unit (IOMMU) is a way to support device pass-through.

On IBM Z, device pass-through is accomplished by using the virtual function I/O (VFIO) device driver, which reserves the pass-through device for KVM guests and accesses the corresponding host resource on behalf of the guest.

3.5 Configuring the KVM host

This section describes how to enable RHEL as a KVM host and set up the devices to be ready for VM guest use.

3.5.1 Defining NICs

As described in 3.1, “Defining the target configuration” on page 48, in our lab environment, we use one NIC through the 1e20-1e22 triplet OSA devices, which is defined in the E2 OSA channel for management purposes. For the VM guest network, we used the MacVTap network that uses a bond interface with two NICs (OSA E2 and OSA E4).

As shown in Example 3-10, the only NIC configured is the one that we used for the RHEL installation.

Example 3-10 Configured networks

```
[root@rdbkvmr ~]# znetconf -c
Device IDs          Type      Card Type      CHPID Drv. Name      State
-----
0.0.1e80,0.0.1e81,0.0.1e82 1731/01 OSD_10GIG      E8 qeth enc1e80      online
```

By following the architecture that is proposed for this lab, we need to add for the guest network two NICs (OSA triplets) with different OSA cards that access the same network through different switches.

Example 3-11 shows two unconfigured NICs that were added with different OSA cards and CHPIDs, which provide redundancy for the virtual environment.

Example 3-11 Checking NICs availability

```
[root@rdbkvm1 network-scripts]# znetconf -u | grep e8
Scanning for network devices...
0.0.1e83,0.0.1e84,0.0.1e85 1731/01 OSA (QDIO)      e8 qeth
0.0.1ee3,0.0.1ee4,0.0.1ee5 1731/01 OSA (QDIO)      ee qeth
```

As shown in Example 3-12, we configure the 0.0.1e83-0.0.1e85 device as interface eth0 and the 0.0.1ee3-0.0.1ee5 device as interface eth1.

Example 3-12 Configuring the NICs

```
[root@rdbkvm1 network-scripts]# znetconf -a 1e83 -o layer2=1 -o
buffer_count=128
Scanning for network devices...
Successfully configured device 0.0.1e83 (enc1e83)
[root@rdbkvm1 network-scripts]# znetconf -a 1ee3 -o layer2=1 -o
buffer_count=128
Scanning for network devices...
Successfully configured device 0.0.1ee3
(enc1ee3)
```

To check whether the configuration is correct, run the command for each NIC that is shown in Example 3-13.

Example 3-13 Validating NICs configuration

```
[root@rdbkvm1 network-scripts]# lsqeth enc1e83
Device name          : enc1e83
-----
card_type            : OSD_10GIG
cdev0                : 0.0.1e83
cdev1                : 0.0.1e84
cdev2                : 0.0.1e85
chpid                : E8
online               : 1
portname             : no portname required
portno               : 0
state                : UP (LAN ONLINE)
priority_queueing    : always queue 2
buffer_count         : 128
layer2               : 1
isolation            : none
bridge_role          : none
```

```
bridge_state           : inactive
bridge_hostnotify     : 0
bridge_reflect_promisc : none
switch_attrs          : unknown
vnicc/flooding         : 0
vnicc/learning         : 0
vnicc/learning_timeout : 600
vnicc/mcast_flooding  : 0
vnicc/rx_bcast         : 1
vnicc/takeover_learning : 0
vnicc/takeover_setvmac : 0
```

3.5.2 Defining the bond interface

To have network high availability (HA), we define a bond interface named `bond1` (master). This interface accesses the physical network through two NIC subordinate interfaces: `enc1e83` and `enc1ee3`. Those interfaces were created in the previous section.

Example 3-14 shows how to define a bond interface and set `enc1e83.008` and `enc1ee3.008` as subordinate interfaces of the `bond1` interface. To change the properties of the NICs, these interfaces must be down.

Example 3-14 Defining a bond interface

```
root@rdbkvm1 network-scripts]# nmcli con add type bond ifname bond1 bond.options
"mode=balance-tlb,miimon=100"
Connection 'bond-bond1' (c5ef5400-04e4-4fb9-bd64-49215c37f227) successfully added
```

If you dedicated OSAs for the subordinates and the suitable configuration for the switches, you can configure the bonding option as `mode=802.3ad miimon=100`, which allows you to create LACP aggregation groups that share the same speed and duplex settings.

If two OSA Express7s 10 Gb are used, you can aggregate two 10 Gb per second (Gbps) ports into a 20 Gbps trunk port. This configuration is equivalent of having one interface with 20 Gbps speed. It provides fault tolerance and load balancing.

As shown in Example 3-14, we verify that the definition of the `bond0` interface is correct.

Example 3-15 Adding subordinates to the master bond bond1

```
[root@rdbkvm1 ~]# nmcli con add type ethernet ifname enc1e83 master bond1
Connection 'bond-slave-enc1e83' (0ca5a7d1-7787-4876-9664-796defd507ba)
successfully added.
[root@rdbkvm1 ~]# nmcli con add type ethernet ifname enc1ee3 master bond1
Connection 'bond-slave-enc1ee3' (8ddaef88-f7f3-4d3c-96af-d2d36aebcae3)
successfully added.
[root@rdbkvm1 ~]# nmcli con mod bond-bond1 ipv4.method disabled
[root@rdbkvm1 ~]# nmcli con mod bond-bond1 ipv6.method disabled
```

After the subordinate interfaces are added (see Example 3-15), some 802-3-ethernet.s390 parameters of the OSA subordinate interfaces (nettype, subchannels, and options) must be updated (see Example 3-16).

Example 3-16 Updating the slave BOND interface parameters related to s390

```
[root@rdbkvm1 ~]# nmcli con mod bond-slave-enc1e83 802-3-ethernet.s390-nettype qeth
[root@rdbkvm1 ~]# nmcli con mod bond-slave-enc1e83 802-3-ethernet.s390-subchannels
0.0.1e83,0.0.1e84,0.0.1e85
[root@rdbkvm1 ~]# nmcli con mod bond-slave-enc1e83 802-3-ethernet.s390-options "layer2=1
buffer_count=128"
[root@rdbkvm1 ~]# nmcli con mod bond-slave-enc1ee3 802-3-ethernet.s390-nettype qeth
[root@rdbkvm1 ~]# nmcli con mod bond-slave-enc1ee3 802-3-ethernet.s390-subchannels
0.0.1ee3,0.0.1ee4,0.0.1ee5
[root@rdbkvm1 ~]# nmcli con mod bond-slave-enc1ee3 802-3-ethernet.s390-options "layer2=1
buffer_count=128"
```

Also, the permanent (persistent) configurations must be fixed by using **vi** to remove the **//** in the **OPTIONS** statements, as shown in the following examples:

- ▶ [root@rdbkvm1 ~]# vi /etc/sysconfig/network-scripts/ifcfg-bond-slave-enc1e83
Remove the **//** in the **OPTIONS** line
- ▶ [root@rdbkvm1 ~]# vi /etc/sysconfig/network-scripts/ifcfg-bond-slave-enc1ee3
Remove the **//** in the **OPTIONS** line

We used the VLAN 8 in our lab; therefore, we must create a VLAN subinterface of the bond1 connection (see Example 3-17).

Example 3-17 Creating the bond1.008 VLAN 8 subinterface

```
[root@rdbkvm1~]# nmcli con add type vlan con-name bond1.008 ifname bond1.008 dev bond1 id 8
[root@rdbkvm1~]# nmcli con mod bond1.008 ipv4.method disabled
[root@rdbkvm1~]# nmcli con mod bond1.008 ipv6.method disabled
```

For more information about VLANs on RHEL, see this Red Hat [web page](#).

Now, we can check our defined connections status (see Example 3-18).

Example 3-18 Checking connections status

```
[root@rdbkvm1 bonding]# nmcli con show
NAME                                UUID                                TYPE    DEVICE
bond1.008                           da6c399e-85f5-413d-a3b3-9437700d7da2  vlan    bond1.008
bond-bond1                           88435ff5-c60a-4859-b9a3-699b40f39c41  bond    bond1
bond-slave-enc1e83                   2d04a5fa-491d-4a08-958c-e3f108c1bfd1  ethernet enc1e83
bond-slave-enc1ee3                   a99c7330-645b-4068-ad46-296c51242a69  ethernet enc1ee3
```

If some connections are not displayed in green, you must start the connection by using the **nmcli con up connection_name** command.

As shown in Example 3-19, we verify that the definition of the bond1 interface is correct.

Example 3-19 Validating bond interface

```
root@rdbkvm1 rhe184]# cat /proc/net/bonding/bond1
Ethernet Channel Bonding Driver: v4.18.0-305.25.1.el8_4.s390x
```

```
Bonding Mode: transmit load balancing
Primary Slave: None
Currently Active Slave: encl83
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0
Peer Notification Delay (ms): 0
```

```
Slave Interface: encl83
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: f2:b1:16:7b:16:be
Slave queue ID: 0
```

```
Slave Interface: encl83
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 5e:7c:1a:ea:68:8c
Slave queue ID: 0
```

For more information about bonding, see the IBM publication [Linux Channel Bonding Best Practices and Recommendations](#).

3.5.3 Defining HiperSocket interfaces

HiperSockets allows memory-to-memory communication between hosts in the same IBM Z platform. HiperSockets avoid the use of external communications by way of NIC and Ethernet switch. This feature eliminates traditional network latency.

For more information about this feature, see “Network connectivity” on page 6.

As described in 3.1, “Defining the target configuration” on page 48, the HiperSocket CHPID is F4. The triplet for the encf00 interface definition is 0F00 - 0F02 in our lab environment.

3.5.4 Defining HiperSocket interface to support VM guest network

We define the encf00 interface on the HiperSocket chpid(f4) to allow VM guest access to the HiperSocket network.

Example 3-20 shows the HiperSocket device availability.

Example 3-20 List of unconfigured HSI devices on F4 CHPID

```
[root@rdbkvm1 dev]# znetconf -u | grep " f4 "
```

0.0.0f00,0.0.0f01,0.0.0f02	1731/05	HiperSockets	f4	qeth
0.0.0f03,0.0.0f04,0.0.0f05	1731/05	HiperSockets	f4	qeth
0.0.0f06,0.0.0f07,0.0.0f08	1731/05	HiperSockets	f4	qeth
0.0.0f09,0.0.0f0a,0.0.0f0b	1731/05	HiperSockets	f4	qeth
0.0.0f0c,0.0.0f0d,0.0.0f0e	1731/05	HiperSockets	f4	qeth
0.0.0f0f,0.0.0f10,0.0.0f11	1731/05	HiperSockets	f4	qeth
0.0.0f12,0.0.0f13,0.0.0f14	1731/05	HiperSockets	f4	qeth
0.0.0f15,0.0.0f16,0.0.0f17	1731/05	HiperSockets	f4	qeth
0.0.0f18,0.0.0f19,0.0.0f1a	1731/05	HiperSockets	f4	qeth
0.0.0f1b,0.0.0f1c,0.0.0f1d	1731/05	HiperSockets	f4	qeth

Choose the 0.0.0f00, 0.0.0f01, and 0.0.0f02 devices to create the encf00 interface, as shown in Example 3-21.

Example 3-21 Configuring the HiperSocket interface

```
[root@rdbkvm1 dev]# znetconf -a 0f00 -o layer2=1 -o buffer_count=128
Scanning for network devices...
Successfully configured device 0.0.0f00 (encf00)
```

To have a persistent definition of the encf00 HiperSocket interface, create the configuration file, as shown in Example 3-22.

Example 3-22 Making HiperSocket changes permanent

```
[root@rdbkvm1 home]# nmcli con add type ethernet ifname encf00 con-name encf00
Connection 'encf00' (bbb079f1-7037-4159-a755-a77ab6ad2a06) successfully added.
[root@rdbkvm1 home]# nmcli con mod encf00 802-3-ethernet.s390-nettype qeth
[root@rdbkvm1 home]# nmcli con mod encf00 802-3-ethernet.s390-subchannels
0.0.0f00,0.0.0f01,0.0.0f02
[root@rdbkvm1 home]# nmcli con mod encf00 802-3-ethernet.s390-options "layer2=1
buffer_count=128"
[root@rdbkvm1 home]# nmcli con mod encf00 ipv4.method disabled
[root@rdbkvm1 home]# nmcli con mod encf00 ipv6.method disabled
```

Also, we must check the permanent configurations that are correcting the OPTIONS statements (see Example 3-23).

Example 3-23 Correcting the encf000 hsi persistent configuration

```
vi /etc/sysconfig/network-scripts/ifcfg-encf00 ---> REMOVE the // in the OPTIONS line
```

The next step is to start and validate the new interface, as shown in Example 3-24.

Example 3-24 HiperSocket interface start and validation

```
[root@rdbkvm1 network-scripts]# nmcli con up encf00
Connection successfully activated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/27)
[root@rdbkvm1 network-scripts]# nmcli con show encf00
connection.id:                encf00
connection.uuid:              bbb079f1-7037-4159-a755-a77ab6ad2a06
connection.stable-id:        --
connection.type:              802-3-ethernet
[...]
802-3-ethernet.port:         --
802-3-ethernet.speed:        0
802-3-ethernet.duplex:       --
802-3-ethernet.auto-negotiate: no
802-3-ethernet.mac-address:  --
802-3-ethernet.cloned-mac-address: --
802-3-ethernet.generate-mac-address-mask:--
802-3-ethernet.mac-address-blacklist: --
802-3-ethernet.mtu:          auto
802-3-ethernet.s390-subchannels: 0.0.0f00,0.0.0f01,0.0.0f02
802-3-ethernet.s390-nettype:  qeth
802-3-ethernet.s390-options:  layer2=1 buffer_count=128
[...]
```

3.5.5 Defining HiperSocket KVM host interface

We also must define a HiperSocket interface for KVM use. To set up this interface, select the 0.0.0f03, 0.0.0f04, and 0.0.0f05 devices to create interface encf03, as shown in Example 3-25.

Example 3-25 Configuring the HiperSocket interface

```
[root@rdbkvm1 network-scripts]# znetconf -a 0f03 -o layer2=1 -o buffer_count=128
Scanning for network devices...
Successfully configured device 0.0.0f03 (encf03)
```

To have a persistent definition of the encf03 HiperSocket interface, we must define the network manager's connection, as shown in Example 3-26.

Example 3-26 Making HiperSocket changes permanent and assigning an IP address

```
[root@rdbkvm1 network-scripts]# nmcli con add type ethernet ifname encf03 con-name encf03
Connection 'encf03' (cb02de05-0a2e-48da-9d3d-6660935e2966) successfully added.
[root@rdbkvm1 network-scripts]# nmcli con mod encf03 802-3-ethernet.s390-nettype qeth
[root@rdbkvm1 network-scripts]# nmcli con mod encf03 802-3-ethernet.s390-subchannels
0.0.0f03,0.0.0f04,0.0.0f05
[root@rdbkvm1 network-scripts]# nmcli con mod encf03 802-3-ethernet.s390-options "layer2=1
buffer_count=128"
[root@rdbkvm1 network-scripts]# nmcli con mod encf03 ipv6.method disabled
[root@rdbkvm1 network-scripts]# nmcli con mod encf03 ipv4.method auto
[root@rdbkvm1 network-scripts]# nmcli con mod encf03 ipv4.addresses 100.150.233.40/24
[root@rdbkvm1 network-scripts]# nmcli con mod encf03 ipv4.method manual
[root@rdbkvm1 network-scripts]# nmcli con up encf03
```

Also, we must check the permanent configurations that are correcting the OPTIONS statements (see Example 3-27).

Example 3-27 Correcting the encf000 HSI persistent configuration

```
vi /etc/sysconfig/network-scripts/ifcfg-encf03 ---> REMOVE the // in OPTIONS line
```

3.5.6 Defining HiperSocket Converged interface

By using HiperSockets Converged Interface (HSCI) connections, a HiperSockets network interface can be combined with an external OSA- or RoCE port, which creates a single network interface. With this interface, we can access the switched network and the intra-CEC HiperSocket network with the same IP. Both of the devices that are participating in the HSCI interface must have the same physical network (PNET) ID.

For our lab, we choose the adapters that are listed in Table 3-4.

Table 3-4 Lab adapters

Device type	CHPID	Devices	PNETID
HiperSocket	F2	0.0.0FC9,0.0.0FCA,0.0.0FCB	PERFNET
OSA Express	EE	0.0.1EE9,0.0.1EEA,0.0.1EEB	PERFNET

The sequence of commands that is used to check the HiperSocket PNET ID device is shown in Example 3-28 - Example 3-32 on page 69.

Example 3-28 Checking the OSA PNET ID

```
[root@rdbkvm1 isos]# cat /sys/devices/css0/chp0.ee/util_string | iconv -f
IBM-1047 -t ASCII
PERFNET
```

Example 3-29 Creating HSI and OSA interfaces

```
[root@rdbkvm1 ~]# znetconf -a 0fc9 -o layer2=1 -o buffer_count=128
Scanning for network devices...
Successfully configured device 0.0.0fc9 (encfc9)
[root@rdbkvm1 ~]# znetconf -a 1ee9 -o layer2=1 -o buffer_count=128
Scanning for network devices...
Successfully configured device 0.0.1ee9 (enclee9)
```

Example 3-30 Checking the HSCI interface

```
[root@rdbkvm1 ~]# hsci add encfc9 enclee9
Verifying net dev enclee9 and HiperSockets dev encfc9
Adding hsci0fc9 with a HiperSockets dev encfc9 and an external dev enclee9
Set encfc9 MAC 0e:00:f2:35:00:0b on enclee9 and hsci0fc9
Successfully added HSCI interface hsci0fc9
```

Example 3-31 Creating the VLAN 8 interface from HSCI 0FC9 device and assign IP

```
[root@rdbkvm1 ~]# ip link add dev hsci0fc9.8 link hsci0fc9 type vlan id 8
[root@rdbkvm1 ~]# ip addr add 129.40.23.232/24 dev hsci0fc9.8
[root@rdbkvm1 ~]# ip link set up hsci0fc9.8
```

Example 3-32 Checking HSCI interface

```
[root@rdbkvm1 isos]# hsci show
HSCI      PNET_ID          HiperSockets      External
-----
hsci0fc9  PERFNET          encfc9            enclee9
```

3.5.7 Defining SMC interfaces

SMC-D and SMC-R use shared memory to provide low-latency, high-bandwidth, and cross-LPAR connections for applications. This support is intended to provide application-transparent direct memory access (DMA) communications to TCP endpoints for socket-based connections.

Installing SMC tools package

To support SMC-D (ISM) and SMC-R (RoCE), you must install the SMC-tools package. For more information about obtaining the packages, see this [GitHub web page](#).

After downloading the content, you must extract the packages and upload them to the host (in our example to /home/isos/).

Use the commands that are shown in Example 3-33 to install the packages.

Example 3-33 Installing SMC

```
[root@rdbkvm1 isos]# yum install libn*
[root@rdbkvm1 smc-tools-main]# make
```

Enabling SMC-D

This section provides the basic commands to enable SMC-D on the RHEL host server.

Example 3-34 shows how to check the ISM device availability.

Example 3-34 Checking PCI devices

```
[root@rdbkvm1 smc-tools-main]# lspci
00:00.0 Non-VGA unclassified device: IBM Internal Shared Memory (ISM) virtual PCI device
```

As described in 3.5.1, “Defining NICs” on page 61 and Example 3-35 and Example 3-36, we check the PNET ID of the ISM device and in the OSA. Both should represent the same PNET ID.

Example 3-35 Checking ISM device PNET ID

```
[root@rdbkvm1 smc-tools-main]# cat
/sys/devices/pci0000:00/0000:00:00.0/util_string | iconv -f IBM-1047 -t ASCII
PERFNET
```

Example 3-36 Checking the OSA PNET ID

```
[root@rdbkvm1 css0]# cat /sys/devices/css0/chp0.ee/util_string | iconv -f
IBM-1047 -t ASCII
PERFNET
```

In our lab, we define a NIC in the CHPID EE (see Example 3-37). For more information, see 3.5.1, “Defining NICs” on page 61.

Example 3-37 Defining OSA, VLAN interface and assign IP

```
[root@rdbkvm1 home]# znetconf -a 1ee6 -o layer2=1 -o buffer_count=128
Connection ' enc1ee6 ' (dca07eda-d307-48c9-992a-272e460d4486) successfully added.
[root@rdbkvm1 home]# nmcli con add type ethernet ifname enc1ee6 con-name enc1ee6
[root@rdbkvm1 home]# nmcli con mod enc1ee6 802-3-ethernet.s390-nettype qeth
[root@rdbkvm1 home]# nmcli con mod enc1ee6 802-3-ethernet.s390-subchannels
0.0.1ee6,0.0.1ee7,0.0.1ee8
[root@rdbkvm1 home]# nmcli con mod enc1ee6 802-3-ethernet.s390-options "layer2=1
buffer_count=128"
[root@rdbkvm1 home]# nmcli con mod enc1ee6 ipv4.method disabled
[root@rdbkvm1 home]# nmcli con mod enc1ee6 ipv6.method disabled
[root@rdbkvm1 home]# nmcli con add type vlan con-name enc1ee6.008 ifname
enc1ee6.008 dev enc1ee6 id 8
Connection 'enc1ee6.008 ' (97433a8e-63d9-40e4-98f8-5b94e1ffbdb4) successfully
added.
[root@rdbkvm1 home]# nmcli con add type vlan con-name enc1ee6.008 ifname
enc1ee6.008 dev enc1ee6 id 8
Connection 'enc1ee6.008 ' (97433a8e-63d9-40e4-98f8-5b94e1ffbdb4) successfully
added.
[root@rdbkvm1 home]# nmcli con mod enc1ee6.008 ipv4.method auto
```

```
[root@rdbkvm1 home]# nmcli con mod encl6e6.008 ipv4.addresses 129.40.23.220/24
[root@rdbkvm1 home]# nmcli con mod encl6e6.008 ipv4.method manual
[root@rdbkvm1 home]# nmcli con mod encl6e6.008 ipv6.method disabled
```

To test the communication between rdbkvm1 and rdbkvm2 LPARs in the same CPC by using the SMC-D, we use the iperf3 tool. To install this tool, run the **yum** command that is in Example 3-38 in each LPAR.

Example 3-38 installing iperf3 tool

```
[root@rdbkvm1 home]# yum -y install iperf3
[root@rdbkvm2 home]# yum -y install iperf3
```

Allow the local firewall (see Example 3-39) to accept connections for iperf3 on the 5201 TCP port on the rdbkvm2 LPAR server.

Example 3-39 Allowing the local firewall port 5201

```
[root@rdbkvm2 home]# firewall-cmd --permanent --add-port=5201/tcp
success
[root@rdbkvm2 home]# firewall-cmd --reload
success
[root@rdbkvm2 ~]# firewall-cmd --list-all
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: bond1 bond1.008 enP6p0s0 enP6p0s0.008 encl6e80 encl6e80.008 encl6e83
encl6ee3 encl6ee6 encl6ee6.008 encf00 encf03
  sources:
  services: cockpit dhcpv6-client ssh
  ports: 21/tcp 5901/tcp 5201/tcp
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

Start iperf3 in listening mode by using the command that is shown in Example 3-40 on rdbkvm2.

Example 3-40 Starting iperf3 in listening mode

```
[root@rdbkvm2 ~]# smc_run iperf3 -s
-----
Server listening on 5201
```

Use the command that is shown in Example 3-41 to open another SSH session against the rdbkvm2 server and print the information about the SMC sockets.

Example 3-41 Checking the SMC listening on port 5201

```
[root@rdbkvm2 ~]# smcss -a
```

State	UID	Inode	Local Address	Peer Address	Intf Mode
LISTEN	00000	0526700	0.0.0.0:5201		

To test the SMC connections, run the **iperf3** command on rdbkvm4 that is shown in Example 3-42 and then, check rdbkvm.

Example 3-42 Running iperf3 client on rdbkvm1 to the server rdbkvm2

```
[root@rdbkvm1 network-scripts]# smc_run iperf3 -c 129.40.23.221 -t 10
Connecting to host 129.40.23.221, port 5201
[ 5] local 129.40.23.89 port 47660 connected to 129.40.23.221 port 5201
[ ID] Interval          Transfer      Bitrate      Retr  Cwnd
[ 5]  0.00-1.00    sec  2.97 GBytes  25.5 Gbits/sec  0  14.1 KBytes
[ 5]  1.00-2.00    sec  3.03 GBytes  26.0 Gbits/sec  0  14.1 KBytes
[ 5]  2.00-3.00    sec  3.06 GBytes  26.3 Gbits/sec  0  14.1 KBytes
[ 5]  3.00-4.00    sec  3.19 GBytes  27.4 Gbits/sec  0  14.1 KBytes
[ 5]  4.00-5.00    sec  3.06 GBytes  26.3 Gbits/sec  0  14.1 KBytes
[ 5]  5.00-6.00    sec  3.21 GBytes  27.6 Gbits/sec  0  14.1 KBytes
[ 5]  6.00-7.00    sec  3.53 GBytes  30.3 Gbits/sec  0  14.1 KBytes
[ 5]  7.00-8.00    sec  3.61 GBytes  31.0 Gbits/sec  0  14.1 KBytes
[ 5]  8.00-9.00    sec  3.60 GBytes  30.9 Gbits/sec  0  14.1 KBytes
[ 5]  9.00-10.00   sec  3.57 GBytes  30.7 Gbits/sec  0  14.1 KBytes
- - - - -
[ ID] Interval          Transfer      Bitrate      Retr
[ 5]  0.00-10.00   sec  32.8 GBytes  28.2 Gbits/sec  0
[ 5]  0.00-10.00   sec  32.8 GBytes  28.2 Gbits/sec
                                sender
                                receiver

iperf Done.
```

If multiple subnets are used in your configuration on the same CPC, you can use ISM. SMC-D is enhanced to remove the same subnet restriction by using SMC-Dv2 (see Example).

Example 3-43 Checking the usage of SMV-D

```
[root@rdbkvm2 ~]# smcss -a
```

State	UID	Inode	Local Address	Peer Address	Intf Mode
ACTIVE	00000	0521467	::ffff:129.40.23...:5201	::ffff:129.40.2...:47660	0000
SMCD					
ACTIVE	00000	0521466	::ffff:129.40.23...:5201	::ffff:129.40.2...:47658	0000
SMCD					
LISTEN	00000	0521464	0.0.0.0:5201		

```
[root@rdbkvm2 ~]# smcss -D
```

State	UID	Inode	Local Address	Peer Address	Intf
Mode	GID	Token	Peer-GID	Peer-Token	Linkid


```
ACTIVE          00000 0521467 ::ffff:129.40.23...:5201 ::ffff:129.40.2...:47660 0000
SMCD 10000facb7f88561 0000090210000000 68000fabb7f88561 0000090310000000 00000400
ACTIVE          00000 0521466 ::ffff:129.40.23...:5201 ::ffff:129.40.2...:47658 0000
SMCD 10000facb7f88561 0000090410000000 68000fabb7f88561 0000090510000000 00000400
```

SMC-R

As described in “Enabling SMC-D” on page 70, SMC also can be enabled between different CPCs by using an RoCE card that allows remote direct memory access (RDMA) over the external network (SMC-R).

Example 3-44 shows how to check the RoCE device availability.

Example 3-44 Checking PCI devices

```
[root@rdbkvm1 network-scripts]# lspci0006:00:00.0 Ethernet controller: Mellanox
Technologies MT27710 Family [ConnectX-4 Lx Virtual Function]
```

In Example 3-36 on page 70, the PNET ID in the OSA card is displayed. Example 3-45 shows the PNET ID in the RoCE device. Both should display the same PNET ID.

Example 3-45 Checking RoCE device PNET ID

```
[[root@rdbkvm1 network-scripts]# cat
/sys/devices/pci0006:00/0006:00:00.0/util_string | iconv -f IBM-1047 -t ASCII
PERFNET
```

We use the same configuration as in the SMC-D configuration. However, in this example, we use RoCE 2 instead of ISM.

Example 3-46 shows a similar example to Example 3-45 on page 73; however, the communication uses SMC-R here.

Example 3-46 Test results

```
[root@rdbkvm2 ~]# smcss -a
State      UID      Inode    Local Address          Peer Address           Intf Mode
ACTIVE     00000   0524210 ::ffff:129.40.23...:5201 ::ffff:129.40.2...:47664 0000 SMCR
ACTIVE     00000   0524209 ::ffff:129.40.23...:5201 ::ffff:129.40.2...:47662 0000 SMCR
LISTEN     00000   0524095 0.0.0.0:5201
[root@rdbkvm2 ~]# smcss -R
State UID      Inode    Local Address          Peer Address
Intf Mode Role          IB-device
Port Linkid GID          Peer-GID
ACTIVE 00000 0524210 ::ffff:129.40.23...:5201 ::ffff:129.40.2...:47664 0000
SMCR SERV m1x5_3
01 01 0000:0000:0000:0000:0000:ffff:8128:17df fe80:0000:0000:0000:13c6:58ef:9ce6:336d
ACTIVE 00000 0524209 ::ffff:129.40.23...:5201 ::ffff:129.40.2...:47662 0000
SMCR SERV m1x5_3
01 01 0000:0000:0000:0000:0000:ffff:8128:17df fe80:0000:0000:0000:13c6:58ef:9ce6:336d
```

If multiple subnets are used between IBM z15 CPCS, SMC Version 2 (SMCv2) allows you to enable multiple IP subnet capability for SMC. The multiple subnet capability is enabled by updates to the underlying networking specifications for RoCE (referred to as *RoCEv2*) and the IBM Z ISM feature (referred to as *ISMv2*) along with updates to the related technologies.

Also, if two or more RoCE cards are used, the SMC driver creates an SMC-R link group that allows load-balancing and high availability after the SMC-R communication is established.

For more information about RoCE, see this [IBM Documentation web page](#).

3.5.8 Defining the MacVTap network

This section describes defining two MacVTap networks: one for OSA and one for HiperSockets.

MacVTap for OSA NICs

Instead of the use of the default network connectivity for the guests network address translation (NAT) connections, we chose MacVTap in bridge mode. This mode allows the guests a direct connection with the specified interface in the MacVTap network.

To configure the MacVTap network, complete the following steps

1. Use the `virsh` command and an XML definition file. Example 3-47 shows our `macvtap-net.xml` network definition file.

Example 3-47 macvtap-net.xml

```
[root@rdbkvm1 network]# cat macvtap-net1.xml
<network>
  <name>macvtap-net1</name>
  <forward mode="bridge">
    <interface dev="bond1.008"/>
  </forward>
</network>
```

Example 3-48 shows the `virsh` command that is used to define a MacVTap network.

Example 3-48 virsh net-define command

```
[root@rdbkvm1 network]# virsh net-define macvtap-net1.xml
Network macvtap-net1 defined from macvtap-net1.xml
```

2. Set MacVTap-net persistence and start the network, as shown in Example 3-49.

Example 3-49 virsh net-autostart and net-start command

```
[root@rdbkvm1 network]# virsh net-autostart macvtap-net1
Network macvtap-net1 marked as autostarted
[root@rdbkvm1 network]# virsh net-start macvtap-net1
Network macvtap-net1 started
```

MacVTap for HiperSocket NIC

The same steps that are used in “MacVTap for OSA NICs” on page 74 are applied to the MacVTap HiperSockets definition. Example 3-50 shows the XML file that was created to define the HiperSockets NIC.

Example 3-50 macvtap-hsi.xml

```
[root@rdbkvm1 network]# cat macvtap-hsi.xml
<network>
  <name>macvtap-hsi</name>
  <forward mode="bridge">
    <interface dev="encf00"/>
  </forward>
</network>
```

Also, define start and autoenable the new macvtap-his1 network, as shown in Example 3-50.

3.5.9 Defining crypto adapters and domains

As explained in 2.2.4, “Encryption considerations” on page 33, the Crypto Express card advantages can be used by the KVM hosts and VM guests.

It is important to check the compatibility list for Crypto Express adapters when RHEL is used before beginning the installation. For more information about supported Crypto Express adapters, see this [IBM Documentation web page](#).

To make the AP cards available to the KVM guests (see “Cryptography” on page 57), you use the VFIO mediated device framework to assign cryptographic adapter resources to the device.

To make this assignment, load the `vfio_ap` device driver by running the `modprobe vfio_ap` command and then, add adapters 0x0 to the device, as shown in Example 3-51 and Example 3-52.

Example 3-51 Enabling vfio_ap permanently (adding modules in /etc/modules)

```
[root@rdbkvm1 ~]# vim /etc/modules-load.d/modules01.conf
[root@rdbkvm1 ~]# cat /etc/modules-load.d/modules01.conf
# Load the below modules at boot
vfio_ap
[root@rdbkvm1 ~]# systemctl restart systemd-modules-load
```

Example 3-52 Preparing crypto usage

```
[root@rdbkvm1 vfio_ccw]# modprobe vfio_ap
```

By default, the `zcrypt` device driver controls all AP queues that are available to a KVM host, which makes them unavailable to guests.

You must free all adapters and domains to assign them to a KVM guest. To do so, issue the command that is shown in Example 3-53.

Example 3-53 Freeing all cards and domains from the KVM host

```
[root@rdbkvm1 vfio_ccw]# echo 0x0 > /sys/bus/ap/apmask
[root@rdbkvm1 vfio_ccw]# echo 0x0 > /sys/bus/ap/aqmask
```

Use the **lszcrypt** command to display information about the crypto adapters, as shown in Example 3-54.

Example 3-54 Verifying crypto cards

```
[root@rdbkvm1 ~]# lszcrypt
CARD.DOMAIN TYPE  MODE          STATUS  REQUESTS
-----
03          CEX6C CCA-Coproc  online    1
06          CEX6C CCA-Coproc  online    0
```

Assign AP queues to KVM. Example 3-55 shows the procedure that is used to assign the two crypto cards (03 and 06) and domain (0x25) to the KVM host.

Example 3-55 Crypto for KVM host

```
[root@rdbkvm1 ~]# echo +0x03 > /sys/bus/ap/apmask
[root@rdbkvm1 ~]# echo +0x06 > /sys/bus/ap/apmask
[root@rdbkvm1 ~]# echo +0x25 > /sys/bus/ap/aqmask
```

Example 3-56 shows verifying the crypto assignment to the KVM host.

Example 3-56 Verifying crypto assignment

```
[root@rdbkvm1 /]# lszcrypt
CARD.DOMAIN TYPE  MODE          STATUS  REQUESTS
-----
03          CEX7C CCA-Coproc  online    2
03.0025     CEX7C CCA-Coproc  online    2
06          CEX7C CCA-Coproc  online    0
06.0025     CEX7C CCA-Coproc  online    0
```

One way to make a permanent configuration of the cryptos is running scripts and the use of `rc.local` to run it at start. Example 3-57 - Example 3-62 on page 78 show how to enable `rc.local` `systemd` to run scripts.

Example 3-57 Checking whether `rc_local` service is running

```
[root@rdbkvm1 ~]# systemctl status rc-local
? rc-local.service - /etc/rc.d/rc.local Compatibility
   Loaded: loaded (/usr/lib/systemd/system/rc-local.service; static; vendor
   preset: disabled)
   Active: inactive (dead)
   Docs: man:systemd-rc-local-generator(8)
```

Example 3-58 Checking de rc.local.service systemd unit

```
[root@rdbkvm1 ~]# vim /etc/systemd/system/rc-local.service
[root@rdbkvm1 ~]# cat /etc/systemd/system/rc-local.service
[Unit]
Description=/etc/rc.local Compatibility
ConditionPathExists=/etc/rc.local

[Service]
Type=forking
ExecStart=/etc/rc.local start
TimeoutSec=0
StandardOutput=tty
RemainAfterExit=yes
SysVStartPriority=99

[Install]
WantedBy=multi-user.target
```

Example 3-59 Creating the rc_local file

```
[root@rdbkvm1 ~]# vim /etc/rc.local
[root@rdbkvm1 ~]# cat /etc/rc.local
#!/bin/sh -e
##
exit 0
[root@rdbkvm1 ~]# chmod -v +x /etc/rc.local
mode of '/etc/rc.local' changed from 0644 (rw-r--r--) to 0755 (rwxr-xr-x)
```

Example 3-60 Enabling and starting the rc_local service

```
root@rdbkvm1 ~]# systemctl enable rc-local
Created symlink /etc/systemd/system/multi-user.target.wants/rc-local.service → /etc/systemd/system/rc-local.service.
[root@rdbkvm1 ~]# systemctl start rc-local
[root@rdbkvm1 ~]# systemctl status rc-local
? rc-local.service - /etc/rc.local Compatibility
   Loaded: loaded (/etc/systemd/system/rc-local.service; enabled; vendor preset: disabled)
   Active: active (exited) since Tue 2021-11-30 11:32:29 CST; 8s ago
   Process: 31978 ExecStart=/etc/rc.local start (code=exited, status=0/SUCCESS)

Nov 30 11:32:29 rdbkvm1 systemd[1]: Starting /etc/rc.local Compatibility...
Nov 30 11:32:29 rdbkvm1 systemd[1]: Started /etc/rc.local Compatibility.
```

Example 3-61 Creating a script for freeing ap and aq queues for crypto enablement

```
[root@rdbkvm1 isos]# vim crypto_enablement.sh
[root@rdbkvm1 isos]# chmod u+x /home/isos/crypto_enablement.sh
[root@rdbkvm1 isos]# cat crypto_enablement.sh
#!/bin/bash
# Freeing ap and aq queues for crypto enablement
echo Preparing crypto environment
echo 0x0 > /sys/bus/ap/apmask
echo 0x0 > /sys/bus/ap/aqmask
echo +0x03 > /sys/bus/ap/apmask
echo +0x06 > /sys/bus/ap/apmask
echo +0x25 > /sys/bus/ap/aqmask
```

Example 3-62 Adding the script to rc_local

```
[root@rdbkvm1 isos]# vim /etc/rc.local
[root@rdbkvm1 isos]# cat /etc/rc.local
#!/bin/bash
# THIS FILE IS ADDED FOR COMPATIBILITY PURPOSES
#
# It is highly advisable to create own systemd services or udev rules
# to run scripts during boot instead of using this file.
#
# In contrast to previous versions due to parallel execution during boot
# this script will NOT be run after all other services.
#
# Please note that you must run 'chmod +x /etc/rc.d/rc.local' to ensure
# that this script will be executed during boot.
touch /var/lock/subsys/local
/home/isos/crypto_enablement.sh
```

Results that are similar to the results that are shown in Example 3-55 on page 76 verify that the AP queues were assigned for KVM use.

Example 3-63 shows how to generate a Universally Unique Identifier (UUID) for the mediated device, create the mediated device, and assign the crypto cards and crypto domains to it (for use and control).

Example 3-63 Generating a UUID for VM guest

```
[root@rdbkvm1 /]# uuidgen
50e3859c-115d-459e-9e3d-69114c3973c5
[root@rdbkvm1 ~]# echo 50e3859c-115d-459e-9e3d-69114c3973c5 >
/sys/devices/vfio_ap/matrix/mdev_supported_types/vfio_ap-passthrough/create
[root@rdbkvm1 ~]# echo 0x03 >
/sys/devices/vfio_ap/matrix/50e3859c-115d-459e-9e3d-69114c3973c5/assign_adapter
[root@rdbkvm1 ~]# echo 0x06 >
/sys/devices/vfio_ap/matrix/50e3859c-115d-459e-9e3d-69114c3973c5/assign_adapter
[root@rdbkvm1 ~]# echo 0x002f >
/sys/devices/vfio_ap/matrix/50e3859c-115d-459e-9e3d-69114c3973c5/assign_domain
```

```
[root@rdbkvm1 ~]# echo 0x002f >
/sys/devices/vfio_ap/matrix/50e3859c-115d-459e-9e3d-69114c3973c5/assign_control_domain
```

The procedure that is shown in Example 3-63 must be done for each domain that is used by a VM. In our lab, we used the domains 74 and 75. Example 3-64 shows how to verify the mediated device crypto assignment.

Example 3-64 Verifying mediated device crypto assignment

```
[root@rdbkvm1 devices]# cat
/sys/devices/vfio_ap/matrix/50e3859c-115d-459e-9e3d-69114c3973c5/matrix
03.002f
06.002f
```

To make the `vfio_ap` persistent, you must install the `mdevctl` package and then, run the commands that are shown in Example 3-65 on page 79 by using the UUID, adapter, and domains that are used for the mediated device.

The `mdevctl` utility is used for managing and persisting devices in the mediated device framework of the Linux kernel. For more information about the `mdevctl` utility, see this [GitHub web page](#).

Example 3-65 Making vfio_ap mediated device persistent

```
[root@rdbkvm1 ~]# mdevctl define --uuid 50e3859c-115d-459e-9e3d-69114c3973c5
--parent matrix --type vfio_ap-passthrough
[root@rdbkvm1 ~]# mdevctl modify --uuid 50e3859c-115d-459e-9e3d-69114c3973c5
--addattr=assign_adapter --value=0x03
[root@rdbkvm1 ~]# mdevctl modify --uuid 50e3859c-115d-459e-9e3d-69114c3973c5
--addattr=assign_adapter --value=0x06
[root@rdbkvm1 ~]# mdevctl modify --uuid 50e3859c-115d-459e-9e3d-69114c3973c5
--addattr=assign_domain --value=0x002f
[root@rdbkvm1 ~]# mdevctl modify --uuid 50e3859c-115d-459e-9e3d-69114c3973c5
--addattr=assign_control_domain --value=0x002f
[root@rdbkvm1 ~]# mdevctl start --uuid 50e3859c-115d-459e-9e3d-69114c3973c5
[root@rdbkvm1 ~]# mdevctl list
50e3859c-115d-459e-9e3d-69114c3973c5 matrix vfio_ap-passthrough (defined)
```

Note: You must start the mediated device after the host is restarted.

3.6 Deploying virtual machines on KVM

In this section, we describe the deployment of VMs in the KVM environment. Although a VM can be created by using various methods, this section describes the use of the `virt-install` command and `virsh` tools.

3.6.1 Creating QCOW2 disk image file

As described in “Disk” on page 55, QCOW2 files are used to create the VM disks.

Example 3-66 shows the command that is used to create a 10-GB QCOW2 file.

Example 3-66 Creating QCOW2 image file

```
[root@rdbkvm1 isos]# qemu-img create -f qcow2 kvm1guest03_vol001.img 10G
Formatting 'kvm1guest03_vol001.img', fmt=qcow2 size=10737418240 cluster_size=65536
lazy_refcounts=off refcount_bits=16
```

3.6.2 Installing a new guest by using virt-install

The `virt-install` command line tool is used for creating VMs on KVM, which uses the libvirt hypervisor management library. Example 3-67 shows how to install a VM by using the `virt-install` command.

Example 3-67 Creating VM guest by using virt-install command

```
[root@rdbkvm1 ~]# virt-install --name kvm1guest03 --memory 4000 --vcpus 2
--os-variant rhe18.4 --disk path=/home/isos/kvm1guest03_vol001.img --network
network:macvtap-net1 --cdrom
/var/ftp/pub/RHEL-8.4.0-20210503.1-s390x-dvd1.iso
```

Consider the following points:

- ▶ The `--name` parameter specifies the name of the VM guest.
- ▶ The `--memory` parameter specifies an amount of RAM that is designated to the VM, which is expressed in megabytes.
- ▶ The `--vcpus` parameter specifies how many vCPUs are assigned to the VM.
- ▶ The `--os-variant` parameter specifies which type of operating system is installed; the option is highly recommended when importing a disk image. If it is not provided, the performance of the created VM is negatively affected. You can see the full list of available operating system by running the `osinfo-query os` command.
- ▶ The `--disk` parameter specifies the media to use as storage for the VM guest; `kvmrvm01` uses QCOW2 files. If the file was preallocated, specify the `--import` parameter. Otherwise, omit the `--import` parameter and insert a new file path by using the parameter's format and size to allocate the file during the installation.
- ▶ The `--network` parameter specifies the network options for the VM guest. In this case, we are connecting the guest to the MacVTap-net that was created as described in 3.5.8, “Defining the MacVTap network” on page 74.
- ▶ For the installation source, we used a `.iso` file that uses the `--cdrom` parameter. You also can install from other sources, such as an FTP server.

After the command that is shown in Example 3-67 on page 80 is issued, the VM installation begins, as shown in Figure 3-6.

```
Starting install...
Connected to domain kvm1guest03
Escape character is ^]
[ 0.398874] random: fast init done
[ 3.625497] Freeing initrd memory: 47172K
[ 3.629251] alg: No test for crc32be (crc32be-vx)
[ 3.630612] hypfs: The hardware system does not support hypfs
[ 3.630620] hypfs: Initialization of hypfs failed with rc=-61
[ 3.630883] Initialise system trusted keyrings
[ 3.630889] Key type blacklist registered
[ 3.630947] workingset: timestamp_bits=42 max_order=20 bucket_order=0
[ 3.632031] pstore: using deflate compression
[ 3.632181] Platform Keyring initialized
[ 3.711534] NET: Registered protocol family 38
[ 3.711540] Key type asymmetric registered
[ 3.711541] Asymmetric key parser 'x509' registered
[ 3.711558] Block layer SCSI generic (bsg) driver version 0.4 loaded (major 249)
[ 3.711615] io scheduler mq-deadline registered
[ 3.711617] io scheduler kyber registered
[ 3.711649] io scheduler bfq registered
[ 3.711740] atomic64_test: passed
[ 3.711865] hvc_iucv: The z/VM IUCV HVC device driver cannot be used without z/VM
[ 3.713204] random: crng init done
[ 3.714278] rdac: device handler registered
[ 3.714330] hp_sw: device handler registered
[ 3.714332] emc: device handler registered
[ 3.714352] alua: device handler registered
[ 3.714437] cio: Channel measurement facility initialized using format extended (mode
autodetected)
[ 3.714600] drop_monitor: Initializing network drop monitor service
[ 3.714663] Initializing XFRM netlink socket
[ 3.714721] NET: Registered protocol family 10
[ 3.714845] sclp_sd: Store Data request failed (eq=2, di=3, response=0x40f0,
flags=0x00, status=0, rc=-5)
[ 3.715056] Segment Routing with IPv6
[ 3.715068] NET: Registered protocol family 17
[ 3.715846] mpls_gso: MPLS GSO support
[ 3.715902] registered taskstats version 1
[ 3.715914] Loading compiled-in X.509 certificates
[ 3.805488] Loaded X.509 cert 'Red Hat Enterprise Linux kernel signing key:
a7692646c01499ee798d1448e7130abb87e6dae9'
[ 3.805926] Loaded X.509 cert 'Red Hat Enterprise Linux Driver Update Program (key
3): bf57f3e87362bc7229d9f465321773dfdf1f77a80'
[ 3.806343] Loaded X.509 cert 'Red Hat Enterprise Linux kpatch signing key:
4d38fd864ebe18c5f0b72e3852e2014c3a676fc8'
[ 3.806353] page_owner is disabled
[ 3.808779] Key type big_key registered
[ 3.808788] ima: No TPM chip found, activating TPM-bypass!
[ 3.808793] ima: Allocated hash algorithm: sha256
[ 3.808800] ima: No architecture policies found
[ 3.809482] Freeing unused kernel memory: 3428K
[ 3.838980] Write protected read-only-after-init data: 44k
[
```

```

[ 3.845012] systemd[1]: systemd 239 (239-45.e18) running in system mode.
(+PAM +AUDIT +SELINUX +IMA -APPARMOR +SMACK +SYSVINIT +UTMP +LIBCRYPTSETUP
+GCRYPT +GNUTLS +ACL +XZ +LZ4 +SECCOMP +BLKID +ELFUTILS +KMOD +IDN2 -IDN +PCRE2
default-hierarchy=legacy)
[ 3.845189] systemd[1]: Detected virtualization kvm.
[ 3.845191] systemd[1]: Detected architecture s390x.
[ 3.845192] systemd[1]: Running in initial RAM disk.

Welcome to Red Hat Enterprise Linux 8.4 (Ootpa) dracut-049-135.git20210121.e18
(Initramfs).

```

Figure 3-6 VM Guest installation process through `virt-install`

For more information about the `virt-install` command, see this [Red Hat web page](#).

3.6.3 Cloning a guest by using `Virsh`

`Virsh` is a command-line program that is used to manage VM guests and the hypervisor. It also uses the `libvirt` hypervisor management library. This section shows how to clone a VM from a previous image installation base.

Example 3-68 shows the first task: copying the QCOW2 file (`kvmrvm01_vol001.img`) to `kvmrvm01_vol002.img`.

Example 3-68 Copying the QCOW2 file

```
[root@rdbkvm1 images]# cp kvmrvm01_vol001.img kvmrvm02_vol001.img
```

Run the `dumpxml` command to return the guest VM's configuration file. As shown in Example 3-69, we obtain the XML configuration file `kvmrvm02.xml` from the VM guest, `kvmrvm01`.

Example 3-69 Creating the guest configuration file.

```
[root@rdbkvm1 images]# virsh dumpxml kvmrvm01 > kvmrvm02.xml
```

Because this VM guest is cloned, you must edit `kvmrvm02.xml` by completing the following steps:

1. Change the VM name in the file from `<name>kvmrvm01</name>` to `<name>kvmrvm02</name>`.
2. Delete the UUID assignment statement:


```
<uuid>251e124e-2295-4126-8944-ae080e26c27e</uuid>
```
3. Change the source file of QCOW2 disk from:


```
<source file='/var/lib/libvirt/images/kvmrvm01_vol001.img' />
```

 to:


```
<source file='/var/lib/libvirt/images/kvmrvm02_vol001.img' />
```
4. In the `<interface type='direct'>` section:
 - a. Delete the MAC address statement: `<mac address='52:54:00:6b:8d:f7' />`.
 - b. Delete target device statement: `<target dev='macvtap1' />`.

All deleted information is dynamically generated when the `virsh define` command is used.

The `kvmrvm02` guest is defined as shown in Example 3-70.

Example 3-70 kvmrvm02 guest definition

```
[root@rdbkvm1 images]# virsh define kvmrvm02.xml
Domain kvm1guest02 defined from kvm1gues02.xml
```

To start the new cloned guest, run a `virsh start kvm1guest02` command.

You must change the basic parameters of the new guest, such as the IP address and host name.

Another method that can be used to clone guest is by using the `virt-clone` command, as show in the Example 3-71. The guest to be cloned *must* be shut down.

Example 3-71 Cloning the kvm1guest01 guest

```
[root@rdbkvm1 isos]# virt-clone --original kvm1guest01 --name kvm1guest02 --file
/home/isos/kvm1guest02_vol001.img
Allocating 'kvm1guest02_vol001.img'
| 10 GB 00:00:01
```

Clone 'kvm1guest02' created successfully.

Consider the following points:

- ▶ The `--original` statement indicates the name of the guest (domain) to be cloned.
- ▶ The `--name` statement indicates the name of the new guest (domain) to be created.
- ▶ The `--file` statement indicates the location of the new `qcow2` that is to be allocated for the new guest.

3.6.4 Adding HiperSockets to the VM guest

To add a NIC, a VM is needed to shut down the guest and edit the domain definition. In our example, a vNIC, `macvtap-hsi` is used, which targets the `encf00` HiperSocket interface.

Example 3-72 shows the command that is used to edit the VM domain definition in XML format.

Example 3-72 Editing domain definition

```
[root@rdbkvm1 images]# virsh edit kvm1guest02
Domain kvm1guest01 XML configuration edited.
```

The definition that is shown in Example 3-73 must be added in the `<devices>` section.

Example 3-73 Interface definition

```
<interface type='network'>
    <source network='macvtap-hsi1'/>
    <model type='virtio'/>
</interface>
```

After the domain starts, the VM shows the new interface, and that the domain definition was updated (see Example 3-74).

Example 3-74 interface verification

At the VM level:

```
[root@kvm1guest02 ~]# ip link
3: enc6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode
DEFAULT group default qlen 1000
    link/ether 52:54:00:f2:74:02 brd ff:ff:ff:ff:ff:ff
```

At the KVM host:

```
[root@rdbkvm1 images]# virsh dumpxml kvm1guest02
[...]
  <interface type='direct'>
    <mac address='52:54:00:f2:74:02' />
    <source network='macvtap-hsi1' portid='50d4217c-5041-439f-99be-d22b08bd08d8'
dev='encf00' mode='bridge' />
    <target dev='macvtap37' />
    <model type='virtio' />
    <alias name='net1' />
    <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0006' />
  </interface>
[...]
```

3.6.5 Adding space to guest from ECKD DASD

To add space to a VM in our lab, we added a full volume DASD as a virtio device. You can also add space by using a logical volume from an LVM pool.

The first step is formatting the volume on the host, as shown in Example 3-75 for the ECKD device 0.0.91A8. For this task, we must check whether the device is available for the LPAR and enable it before the formatting process.

Example 3-75 DASD formatting

```
[root@rdbkvm1 by-path]# lsdasd -a | grep 91a8
0.0.91a8  offline
[root@rdbkvm1 ~]# chccwdev -e 91A8
Setting device 0.0.91a8 online
[root@rdbkvm1 by-path]# dasdfmt -b 4096 /dev/disk/by-path/ccw-0.0.91a8 -p
--label=0x91A8
Drive Geometry: 60102 Cylinders * 15 Heads = 901530 Tracks
Device Type: Thinly Provisioned
```

Format the device `/dev/disk/by-path/ccw-0.0.91a8` as shown in the following example:

```
Device number of device : 0x91a8
Labelling device       : yes
Disk label            : VOL1
Disk identifier       : 0X91A8
Extent start (trk no) : 0
Extent end (trk no)  : 1
Compatible Disk Layout : yes
Blocksize            : 4096
Mode                 : Quick
Full Space Release   : yes
```

WARNING:

Disk `/dev/disk/by-path/ccw-0.0.91a8` is online on operating system instances in 15 different LPARs.

Ensure that the disk is not being used by a system outside your LPAR.

Note: Your installation might include z/VM systems that are configured to automatically vary on disks, regardless of whether they are subsequently used.

--->> ATTENTION! <<---

All data of that device will be lost.

Type "yes" to continue, no will leave the disk untouched: yes

Releasing space for the entire device...

Skipping format check due to thin-provisioned device.

Formatting the first two tracks of the device.

Finished formatting the device.

Rereading the partition table... ok

To realize more I/O performance on the virtual block devices, we can configure one or more I/O threads for the virtual server and each virtual block device can use one of these I/O threads (see Example 3-76).

Example 3-76 Creating I/O thread for guest

```
[root@rdbkvm1 isos]# virsh iothreadadd --domain kvm1guest01 --id 1 --live
[root@rdbkvm1 isos]# virsh iothreadadd --domain kvm1guest01 --id 1 --confi
```

Now, we can define and attach the new formatted DASD to the guest. Remember to always format the DASD on the host, but create the partitions at the guest level if you plan to assign the entire disk to the guest (see Example 3-77).

Example 3-77 Defining the virtual block device .xml and attach it to the guest

```
[root@rdbkvm1 isos]# vim kvm1guest01_dasd01.xml
[root@rdbkvm1 isos]# cat kvm1guest01_dasd01.xml
<disk type="block" device="disk">
  <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
  <source dev="/dev/disk/by-path/ccw-0.0.91a8"/>
  <target dev="vdb" bus="virtio"/>
  <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x00a8"/>
</disk>
```

```
[root@rdbkvm1 isos]# virsh attach-device kvm1guest01 kvm1guest01_dasd01.xml
--persistent
Device attached successfully
```

In KVM, we can verify the usage of the virtio device, as shown in Example 3-78.

Example 3-78 Verification commands

```
[root@rdbkvm1 isos]# virsh domblklist kvm1guest01
Target  Source
-----
vda     /var/lib/libvirt/images/kvm1guest01.qcow2
vdb     /dev/disk/by-path/ccw-0.0.91a8
sda     -
```

Example 3-51 On the guest, we verify the device availability:

```
root@kvm1guest1:/home/rdbkuser1# lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
loop0                               7:0      0  53.3M  1 loop /snap/core20/1167
loop1                               7:1      0  53.3M  1 loop /snap/core20/1240
loop2                               7:2      0  39.2M  1 loop /snap/snapd/14057
loop3                               7:3      0  30.2M  1 loop /snap/snapd/13639
loop4                               7:4      0  64.7M  1 loop /snap/lxd/21898
loop5                               7:5      0   65M  1 loop /snap/lxd/21620
sr0                                 11:0     1 1024M  0 rom
vda                                 252:0     0   10G  0 disk
  ··vda1                            252:1     0    1G  0 part /boot
  ··vda2                            252:2     0    9G  0 part
    ··ubuntu--vg-ubuntu--lv        253:0     0    9G  0 lvm /
vdb                                252:16    0  41.3G  0 disk
```

3.6.6 Adding DASD space to guest as a VFIO device

Another way to add DASD to the guest is by using a VFIO pass-through device. This device allows the guest to control the entire DASD as a direct device. For this process, you must bring the DASD's subchannel under control of the `vfio_ccw` device driver, create a mediated device for the DASD and then, assign the mediated device to the guest.

This process is shown in Example 3-79 - Example 3-85 on page 88).

Example 3-79 Checking the device subchannel

```
[root@rdbkvm1 ~]# lscss -a | grep 91a9
Device  Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
-----
0.0.91a9 0.0.2647  3390/0c 3990/e9      f0 f0 7f  50525153 00000000
```

Example 3-80 Formatting the DASD device

```
[root@rdbkvm1 by-path]# dasdfmt -b 4096 /dev/disk/by-path/ccw-0.0.91a9 -p
--label=0x91A9
```

Example 3-81 Unbinding the device from host and creating the mediated device for DASD

```
[root@rdbkvm1 network-scripts]# echo 0.0.91a9 >
/sys/bus/ccw/drivers/dasd-eckd/unbind
[root@rdbkvm1 network-scripts]# echo 0.0.2647 >
/sys/bus/css/devices/0.0.2647/driver/unbind
[root@rdbkvm1 network-scripts]# echo 0.0.2647 >
/sys/bus/css/drivers/vfio_ccw/bind
[root@rdbkvm1 network-scripts]# uuidgen
4d56cc2f-f614-4549-934d-656eef9926f5
[root@rdbkvm1 network-scripts]# echo 4d56cc2f-f614-4549-934d-656eef9926f5>
/sys/bus/css/devices/0.0.2647/mdev_supported_types/vfio_ccw-io/create
```

Example 3-82 Adding the mediated device to the definition in the device section

```
[root@rdbkvm1 by-path]# virsh edit kvm1guest01
<hostdev mode="subsystem" type="mdev" model="vfio-ccw">
  <source>
    <address uuid="4d56cc2f-f614-4549-934d-656eef9926f5"/>
  </source>
  <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x00b1"/>
</hostdev>
```

Example 3-83 Checking at the guest level

```
root@kvm1guest1:/home/rdbkuser1# lscss
Device  Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
-----
0.0.0003 0.0.0000  0000/00 3832/08 yes  80  80  ff  00000000 00000000
0.0.0004 0.0.0001  0000/00 3832/03 yes  80  80  ff  00000000 00000000
0.0.0000 0.0.0002  0000/00 3832/02 yes  80  80  ff  00000000 00000000
0.0.0001 0.0.0003  0000/00 3832/01 yes  80  80  ff  00000000 00000000
0.0.0009 0.0.0004  0000/00 3832/01 yes  80  80  ff  00000000 00000000
0.0.0005 0.0.0005  0000/00 3832/12 yes  80  80  ff  00000000 00000000
0.0.0006 0.0.0006  0000/00 3832/12 yes  80  80  ff  00000000 00000000
0.0.0002 0.0.0007  0000/00 3832/10 yes  80  80  ff  00000000 00000000
0.0.00b1 0.0.0008  3390/0c 3990/e9    f0 f0 7f  50525153 00000000
0.0.0007 0.0.0009  0000/00 3832/05 yes  80  80  ff  00000000 00000000
0.0.08.000a 0000/00 3832/04 yes  80  80  ff  00000000 00000000
```

Example 3-84 Enabling the DASD on the guest

```
root@kvm1guest1:/home/rdbkuser1# chzdev -e dasd 0.0.00b1
ECKD DASD 0.0.00b1 configured
Note: The initial RAM-disk must be updated for these changes to take effect:
    - ECKD DASD 0.0.00b1
update-initramfs: Generating /boot/initrd.img-5.13.0-22-generic
Using config file '/etc/zipl.conf'
Building bootmap in '/boot'
Adding IPL section 'ubuntu' (default)
Preparing boot device: vda (0000).
Done.
```

Example 3-85 Verifying the device availability at guest level

```
root@kvm1guest1:/# lsdasd
Bus-ID      Status    Name      Device  Type          BlkSz  Size      Blocks
=====
0.0.00b1   active   dasda     94:0    ECKD (ESE)    4096   42259MB   10818360
root@kvm1guest1:/home/rdbkuser1# lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
loop1                                7:1    0 53.3M  1 loop /snap/core20/1240
loop2                                7:2    0 39.2M  1 loop /snap/snapd/14057
loop3                                7:3    0  65M   1 loop /snap/lxd/21620
loop4                                7:4    0 30.2M  1 loop /snap/snapd/13639
loop5                                7:5    0 64.7M  1 loop /snap/lxd/21898
loop6                                7:6    0 53.4M  1 loop /snap/core20/1272
sr0                                  11:0    1 1024M  0 rom
dasda                                94:0    0 41.3G  0 disk
··dasda1                             94:1    0 41.3G  0 part
  active    dasda    94:0    ECKD (ESE)  4096  42259MB  10818360

root@kvm1guest1:/home/rdbkuser1# lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0                                  11:0    1 1024M  0 rom
dasda                                94:0    0 41.3G  0 disk
··dasda1                             94:1    0 41.3G  0 part
```

To make the `vfio_ccw` persistent, install the `mdevctl` and `driverctl` packages and then, run the commands that are shown in Example 3-86 on page 89 by using the UUID and the subchannel that are selected for this device.

Notes: Consider the following points:

- ▶ Mediated devices can be configured manually by using `sysfs` operations.
- ▶ The `mdevctl` utility is used for managing and persisting devices in the mediated device framework of the Linux kernel. For more information, see this [GitHub web page](#).
- ▶ The `driverctl` device driver is control utility for Linux. For information, see this [GitHub web page](#).

Example 3-86 Making vfio_ccw mediated device persistent

```
[root@rdbkvm1 ~]# driverctl -b css set-override 0.0.2647 vfio_ccw
[root@rdbkvm1 ~]# mdevctl define -u 4d56cc2f-f614-4549-934d-656eef9926f5 -p
0.0.2647 -t vfio_ccw-io -a
[root@rdbkvm1 ~]# mdevctl start --uuid 4d56cc2f-f614-4549-934d-656eef9926f5
[root@rdbkvm1 ~]# mdevctl list
4d56cc2f-f614-4549-934d-656eef9926f5 0.0.2647 vfio_ccw-io (defined)
```

3.6.7 Adding LUNs if FCP SCSI storage is used

To add space to a VM, you can map a target LUN. In this case, you can choose an available LUN to identify the device-ID that are presented to the VM.

As described in 2.2.2, “Storage considerations” on page 27, the following options are available:

- ▶ Entire disk (LUN or ECKD DASD)
- ▶ Partition of the disk
- ▶ Logical volume

For our lab environment, we choose the entire disk.

It is important to map the device-ID by using the multipath ID. This mapping can be achieved in some installations by multipath-friendly names, such as mpathX. To be read by VM migrations, the recommendation is to avoid the use of multipath-friendly names.

Example 3-87 shows how to identify the target LUN.

Example 3-87 Identifying the target LUN

```
[root@rdbkvm1 by-id]# multipath -ll | grep 450000000000000007
mpathc (36005076309ffd145000000000000007) dm-4 IBM,2107900
```

Example 3-88 shows the identification by device ID.

Example 3-88 Device mapper mpath identification by device ID

```
[root@rdbkvm1 by-id]# ls | grep 36005076309ffd145000000000000007
dm-uuid-mpath-36005076309ffd145000000000000007
scsi-36005076309ffd145000000000000007
```

After identifying the target LUN and the device ID for our lab environment, the target disk is:
/dev/disk/by-id/dm-uuid-mpath-36005076309ffd145000000000000007

The next step is to create an XML file to attach the disk with this information, as shown in Example 3-89.

Example 3-89 Device mapper mpath identification by ID

```
[root@rdbkvm1 images]# vim kvm1guest01_block1.xml
[root@rdbkvm1 images]# cat kvm1guest01_block1.xml
  <disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native"/>
    <source
dev="/dev/disk/by-id/dm-uuid-mpath-36005076309ffd1450000000000000007"/>
    <target dev="vdb" bus="virtio"/>
  </disk>
```

Define the disk to the VM guest, as shown in Example 3-90.

Example 3-90 Attaching disk to kvm1guest01 guest

```
[root@rdbkvm1 images]# virsh attach-device kvm1guest01 kvm1guest01_block1.xml
--persistent
Device attached successfully
```

Validate the host and the guest, as shown in Example 3-91.

Example 3-91 Verifying that the host and guest are attached to the disk

From KVM host.

```
[root@rdbkvm1 images]# virsh domblklist kvm1guest01
Target      Source
-----
vda         /var/lib/libvirt/images/kvm1guest01_vol001.img
vdb         /dev/disk/by-id/dm-uuid-mpath-36005076309ffd1450000000000000007
```

From kvm1guest01 guest.

```
[root@kvm1guest01 ~]# lsblk
NAME                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0                  11:0    1 1024M  0 rom
vda                  252:0    0   10G  0 disk
  ··vda1              252:1    0    1G  0 part /boot
  ··vda2              252:2    0    9G  0 part
    ··rhel-root      253:0    0    8G  0 lvm  /
    ··rhel-swap      253:1    0    1G  0 lvm  [SWAP]
vdb                  252:16   0   40G  0 disk
```

3.6.8 Adding cryptography support to the VM guest

In 3.5.9, “Defining crypto adapters and domains” on page 75, the crypto adapters and domain were defined. The AP queues were then assigned for use by KVM. The `vfiio_ap` mediated device was created to enable the assignment of the crypto device to a VM guest.

Complete the following steps to add cryptography support to the VM guest:

1. In the VM domain definition, edit the XML file (see Example 3-92), locate the `<devices>` section, and add the `<hostdev>` section, as shown in Example 3-93.

Example 3-92 Editing VM definitions using virsh

```
[root@rdbkvm1 images]# virsh edit kvm1guest01
Domain kvm1guest01 XML configuration edited
```

Example 3-93 Mediated device definition

```
<hostdev mode='subsystem' type='mdev' managed='no' model='vfiio-ap'>
  <source>
    <address uuid='50e3859c-115d-459e-9e3d-69114c3973c5' />
  </source>
</hostdev>
```

2. Use the true random number generator (TRNG) feature to generate random numbers. Enable this feature by following Example 3-94. For more information, see Chapter 2, “Planning for the Kernel-based Virtual Machine host and guest” on page 21.

Example 3-94 Statement to use TRNG

```
<rng model='virtio'>
  <backend model='random'>/dev/trng</backend>
</rng>
```

3. Recycle the VM and verify the definitions by running the commands that are shown in Example 3-95.

Example 3-95 Verification commands

In KVM, we verify the usage of TRNG:

```
[root@rdbkvm1 ~]# cat /sys/devices/virtual/misc/trng/byte_counter
trng: 16
hwrng: 528
arch: 21240904
total: 212414480
```

On the guest, we verify the crypto availability:

```
root@kvm1guest1:/home/rdbkuser1# lszcrypt
CARD.DOMAIN TYPE MODE STATUS REQUESTS
-----
03          CEX7C CCA-Coproc online      1
03.002f     CEX7C CCA-Coproc online      1
06          CEX7C CCA-Coproc online      0
06.002f     CEX7C CCA-Coproc online      0
```

Upon completion of these steps, the crypto card is available to be used in the entire environment, including the KVM host and the VMs.

For more information, see *Configuring Crypto Express Adapters for KVM Guests*, [SC34-7717](#).

3.6.9 Using the Integrated Accelerator for zEnterprise Data Compression

The Integrated Accelerator for zEnterprise Data Compression (zEDC) with the IBM® z15™ replaces the zEDC Express adapter with on-chip compression. It provides increased throughput and capacity and reduces the cost of storing, processing, and transporting data.

The acceleration with the on-chip Integrated Accelerator for zEDC is available to applications that use zlib or gzip in user spaces and to the kernel zlib.

To check whether your platform can use the Integrated zEDC, you must confirm that the `df1t` feature is available by using the command that is shown in Example 3-96.

Example 3-96 Checking df1t feature

```
[root@kvm1guest03 home]# lscpu | grep df1t
Flags: esan3 zarch stfle msa ldisp eimm dfp edat etf3eh highgprs te vx vxd vxe gs
vxe2 vxp sort df1t
```

In our lab, we have a z15; therefore, the `df1t` feature is available. To use this feature, you must update the `DFLTCC_LEVEL_MASK` environmental variable, which establishes the compression level.

You can update this environmental variable for a command, a session, or for the entire system. For this example, we show at session level that uses a 5.8 Gb text file that is called `operlog.txt`.

By using higher levels of compression, you can save almost 30x of CPU time and 23x of elapsed time. However, the ratio compression can be lower (in our example, by only 2.7%), as shown in Table 3-5 and Table 3-6 on page 93.

Table 3-5 Compression statistics

Compression level	Compression ratio	Elapsed time	Total CPU time
0x0000	91.6%	1m23.855s	1m23.496s
0x0002	91.6%	1m23.708s	1m23.496s
0x007E	88.9%	0m3.415s	0m2.515s
0x01FF	88.9%	0m3.659s	0m2.557s

Table 3-6 Compression exercises

```
root@kvm1guest1:/home/rdbkuser1# export DFLTCC_LEVEL_MASK=0x0000
root@kvm1guest1:/home/rdbkuser1# time gzip -v -c operlog.txt > operlog1_nohw.gz
operlog.txt:  91.6%
real  1m23.855s
user  1m21.341s
sys   0m2.155s
```

```
root@kvm1guest1:/home/rdbkuser1# export DFLTCC_LEVEL_MASK=0x0002
root@kvm1guest1:/home/rdbkuser1# time gzip -v -c operlog.txt > operlog1_lvl_0x0002.gz
operlog.txt:  91.6%
real  1m23.708s
user  1m21.131s
sys   0m2.244s
```

```
root@kvm1guest1:/home/rdbkuser1# export DFLTCC_LEVEL_MASK=0x007e
root@kvm1guest1:/home/rdbkuser1# time gzip -v -c operlog.txt > operlog1_lvl_0x007e.gz
operlog.txt:  88.9%
real  0m3.415s
user  0m0.447s
sys   0m2.068s
```

```
root@kvm1guest1:/home/rdbkuser1# export DFLTCC_LEVEL_MASK=0x01ff
root@kvm1guest1:/home/rdbkuser1# time gzip -v -c operlog.txt > operlog1_lvl_0x01ff.gz
operlog.txt:  88.9%
real  0m3.659s
user  0m0.450s
sys   0m2.107s
```

For more information about Integrated Accelerator for zEDC, see this IBM Support [Dweb page](#).



Preparing the SLES Kernel-based Virtual Machine environment for virtual machine use

This chapter provides instructions to perform an installation of SUSE Linux Enterprise Server (SLES) on an LPAR, prepare it as a Kernel-based Virtual Machine (KVM) host, and deploy KVM guests.

This chapter includes the following topics:

- ▶ 4.1, “Defining the target configuration” on page 96
- ▶ 4.2, “Preparing the infrastructure” on page 98
- ▶ 4.3, “Collecting information” on page 102
- ▶ 4.4, “Installing SUSE on an LPAR as a KVM host” on page 106
- ▶ 4.5, “Preparing the host for virtualization” on page 108
- ▶ 4.6, “Configuring the KVM host” on page 111
- ▶ 4.7, “Deploying VMs on KVM” on page 127

4.1 Defining the target configuration

To prepare the environment for the workloads that runs in the virtual machines (VMs), it is recommended to build an installation plan. For more information, see Chapter 2, “Planning for the Kernel-based Virtual Machine host and guest” on page 21, which includes the requirements for the installation.

This section provides the instructions to configure and deploy a basic KVM environment on SLES15 SP3.

4.1.1 Logical View

The Logical View of our lab environment that is used in this book is shown in Figure 4-1. This view provides an overview of the entire environment and can be built during the planning phase. For more information, see Chapter 2, “Planning for the Kernel-based Virtual Machine host and guest” on page 21.

The following types of networks are available for guests:

- ▶ External network through the MacVTap network (MacVTap-net)
- ▶ Internal IBM Z platform network through the HiperSocket MacVTap network

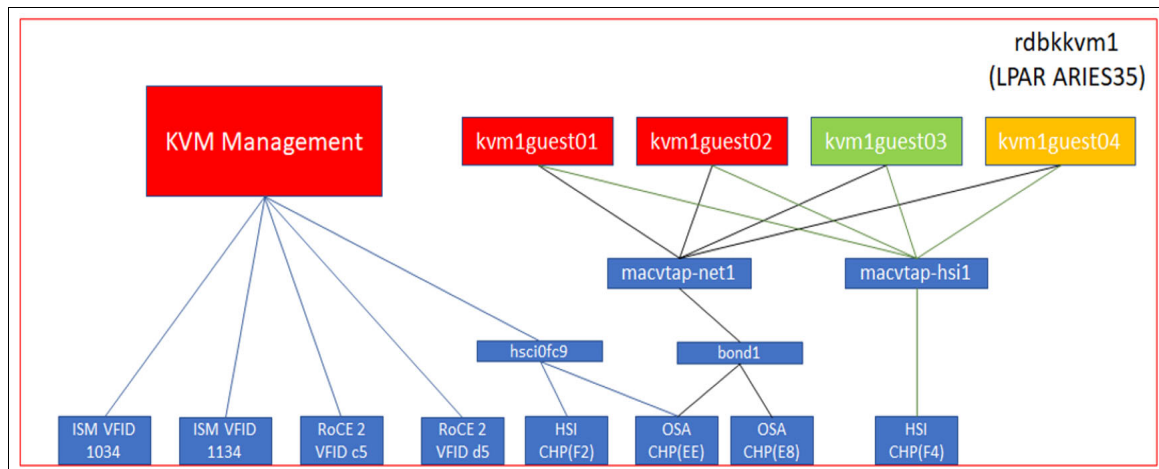


Figure 4-1 SLES Logical View

The KVM host can access the following networks:

- ▶ HiperSockets network through an HSI CHPID(f2) interface.
- ▶ Internal Shared Memory or ISM (SMC-D VFID 1036 and 1136), as described in Chapter 2, “Planning for the Kernel-based Virtual Machine host and guest” on page 21.
- ▶ RoCE network (SMC-R VFID c7 and d7), as described in Chapter 2, “Planning for the Kernel-based Virtual Machine host and guest” on page 21.
- ▶ External network through the OSA network interface card (NIC).

4.1.2 Physical resources

Figure 4-2 shows the following hardware and connectivity setup:

- ▶ One IBM z15 platform with four logical partitions (LPARs)
- ▶ Two OSA adapters that are connected to LAN network
- ▶ Two FICON adapters for connectivity to storage: SCSI devices (FICON as FCP adapter)
- ▶ Four FICON Express16SA+ for connection to the ECKD DASD on IBM DS8900F storage box.
- ▶ One FTP server
- ▶ Two HiperSocket defined CHIPDs
- ▶ Two Crypto Express cards

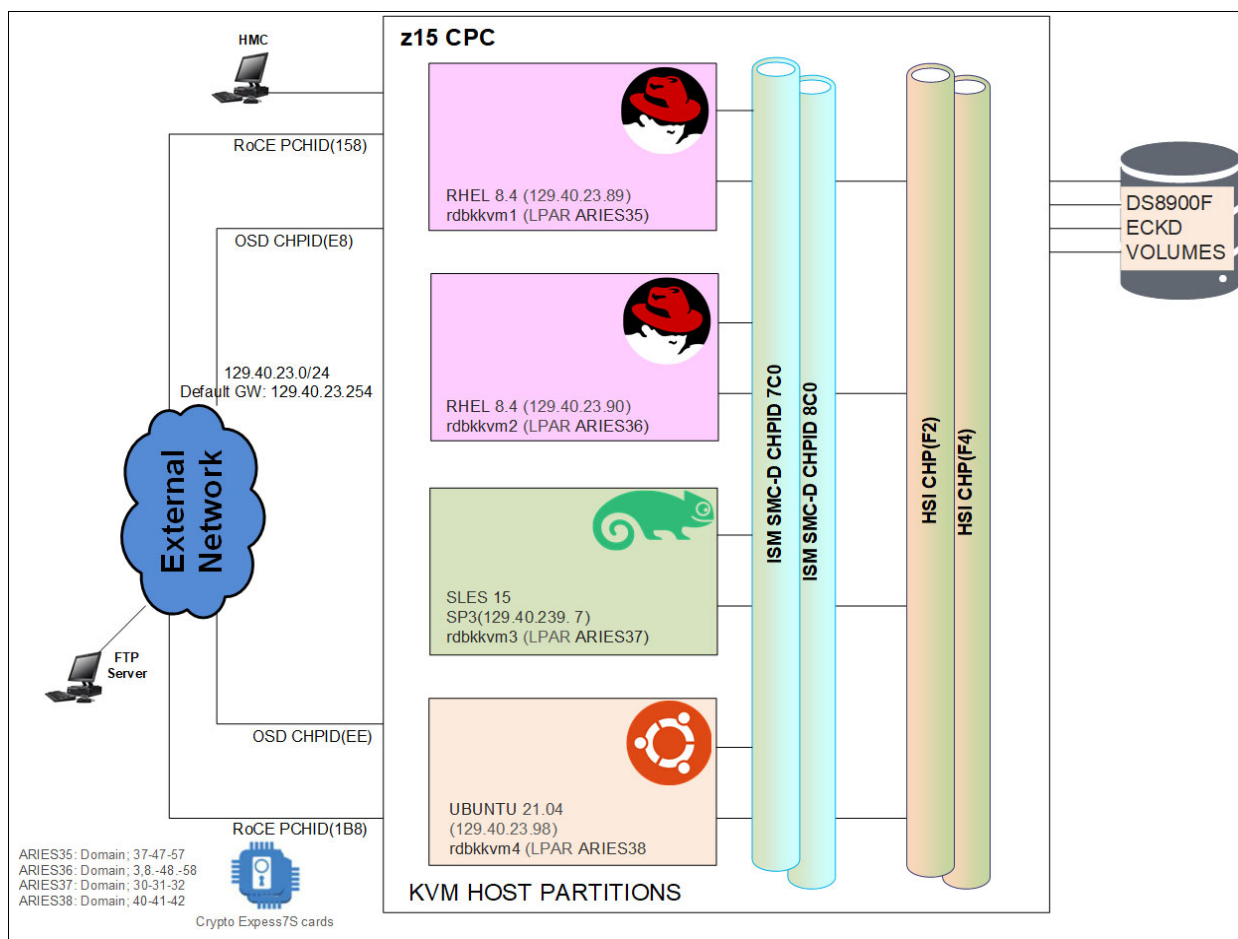


Figure 4-2 SLES physical resources

All LPARs can access all resources. Our lab environment includes the following LPARS:

- ▶ ARIES35: For RHEL
- ▶ ARIES36: For RHEL
- ▶ ARIES37: For SLES
- ▶ ARIES38: For Ubuntu

This chapter is focused on the ARIES37 LPAR for the SLES implementation.

4.1.3 Software resources

For our configuration, we choose SLES 15 SP3, which is the latest supported version for IBM Z. The operating system architecture of the Z platform is s390x and the Linux packages must be based on this architecture.

For more information about SLES-supported versions on IBM Z, see this [IBM Documentation web page](#).

For KVM virtualization (beyond the operating system), the virtualization package is required for the KVM host. For more information, see this [SLES Documentation web page](#).

4.2 Preparing the infrastructure

The IT infrastructure planning depends on many of the factors that are discussed in Chapter 2, “Planning for the Kernel-based Virtual Machine host and guest” on page 21. During the planning phase (see 2.2, “Planning resources for KVM guests” on page 26), we made some decisions about the IT resources that are needed for our lab environment. This section discusses the decisions that we made.

Configuring the resources

For this book, we used the Hardware Management Console (HMC) and Input/Output configuration data set (IOCDs) to set up the resources. For more information about IOCDs, see *I/O Configuration Using z/OS HCD and HCM*, [SG24-7804](#).

For users not familiar with HCD and HMC, the use of Dynamic Partition Manager (DPM) is recommended. For more information, see IBM Support’s [IBM Dynamic Partition Manager Guide](#).

Configuring the storage resources

In our lab configuration, we decided to use ECKD DASD configuration as storage devices for the KVM and the guest storages. You also can use SCSI LUNs that use the Fibre Channel Protocol (FCP) configuration, as described in 2.2.2, “Storage considerations” on page 27.

On IBM Z, the ECKD disk is accessed by using its device address. After the device is formatted under Linux, a volume name that uses the `dasd` prefix and a suffix ranging from `a` - `z` are associated to it (see Table 4-1).

Table 4-1 Storage resources

Device address	Volume name	Capacity	Description
90DE	dasda	400 GB	Rdbkvm3 boot and root disk.
904B	dasdb	54 GB	volume group for kvm guest qcow2 files
914B	dasdc	54 GB	VFIO dasd for kvm3guest3

If you have FCP SCSI LUNs environment, you must work with the your storage team to prepare the disks. The worldwide port name (WWPN) must be given to the storage team for the correct SAN zoning configuration. An example of WWPN information that is needed for the zoning is the WWPN of the IBM Z FCP channels and the storage target ports, as shown in the following example:

- ▶ FCP subchannels WWPN:
 - LUN : 4000400A00000000
 - FCP : B909 WWPN : C05076D08001DA24
 - FCP : C909 WWPN : C05076D0800092A4
- ▶ Storage target PORTS:
 - 5005076309141145: WWPN for P1 storage device port
 - 5005076309149145: WWPN for P2 storage device port
 - 50050763091b1145: WWPN for P3 storage device port
 - 50050763091b9145: WWPN for P4 storage device port

Figure 4-3 shows the SAN configuration for SLES LPAR (ARIES19).

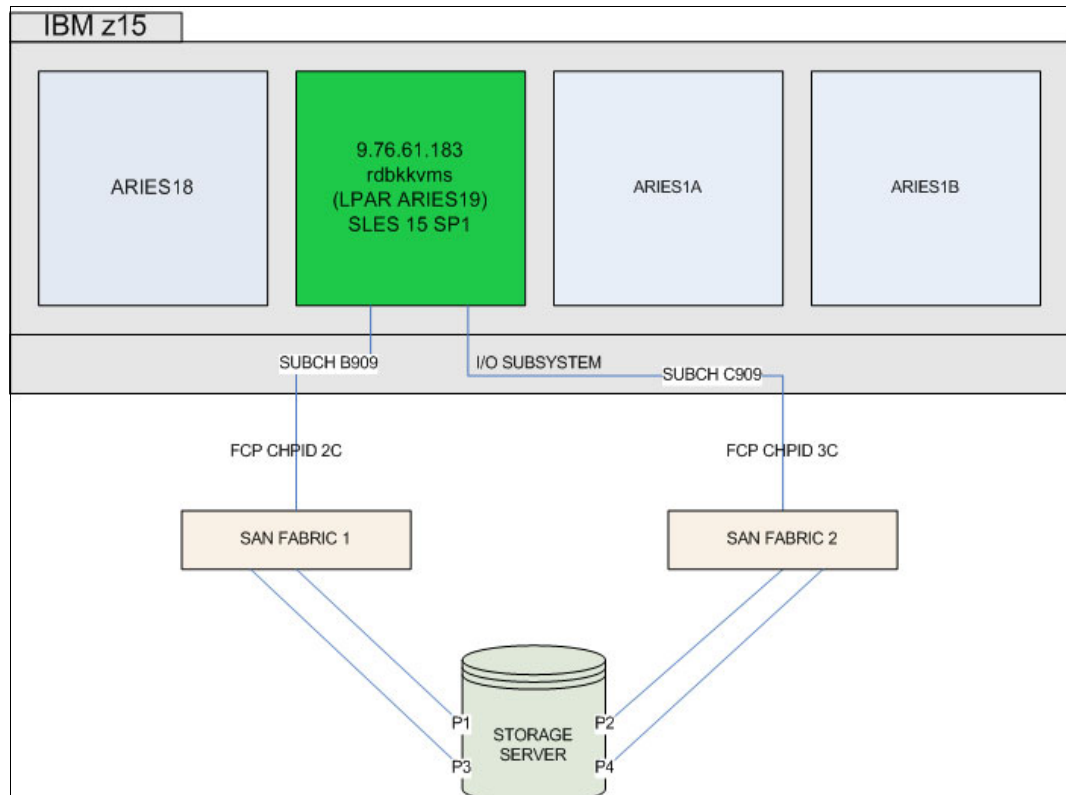


Figure 4-3 SLES SAN configuration example

In this example, we followed the instructions that are described in SUSE’s [Deployment Guide: SUSE Linux Enterprise Server 15 SP3](#). On our FTP server that included an IP address of rdbkftp1.pbm.ihost.com, we created a directory for each ISO file that was downloaded from the SUSE portal and uploaded the contents to the FTP server.

After all of the .ISO files are available on your FTP server, and the server is accessible by the target (HMC or DPM consoles), you can install the host operating system by choosing the FTP method of installation.

FTP can provide a secondary function, which provides access to the local packages repository. You can start the installation process because the first .ISO installer is the only .ISO. The following files are required for a SUSE installation along with the SLES15SP1DVD1/ directory structure:

- ▶ /boot:
 - /s390x (the rdbkkvm3.p parameter file is in this directory)
 - /x86_64
- ▶ /[BOOT]
- ▶ /s390x
- ▶ /repodata
- ▶ /noarch
- ▶ /media.1
- ▶ /docu
- ▶ ARCHIVES.gz
- ▶ CHECKSUMS
- ▶ CHECKSUMS.asc
- ▶ COPYRIGHT
- ▶ COPYRIGHT.de
- ▶ ChangeLog
- ▶ INDEX.gz
- ▶ README
- ▶ gpg-pubkey-307e3d54-5aaa90a5.asc
- ▶ gpg-pubkey-39db7c82-5847eb1f.asc
- ▶ gpg-pubkey-50a3dd1c-50f35137.asc
- ▶ ls-IR.gz
- ▶ rdbkkvm1.ins
- ▶ suse.ins
- ▶ suse_ptf_key.asc
- ▶ Susehmc.ins
- ▶ ARCHIVES.gz
- ▶ CHECKSUMS
- ▶ CHECKSUMS.asc
- ▶ COPYRIGHT
- ▶ COPYRIGHT.de
- ▶ ChangeLog
- ▶ INDEX.gz
- ▶ Module-Basesystem
- ▶ Module-Containers
- ▶ Module-Desktop-Applications
- ▶ Module-Development-Tools

- ▶ Module-Legacy
- ▶ Module-Live-Patching
- ▶ Module-Public-Cloud
- ▶ Module-Python2
- ▶ Module-SAP-Applications
- ▶ Module-Server-Applications
- ▶ Module-Transactional-Server
- ▶ Module-Web-Scripting
- ▶ Product-HA
- ▶ Product-SLES
- ▶ Product-SUSE-Manager-Server-4.2
- ▶ README
- ▶ Boot: /s390x (the prepared rdbkvm3.p parameter file is in this directory)
- ▶ docu
- ▶ glump
- ▶ gpg-pubkey-307e3d54-5aaa90a5.asc
- ▶ gpg-pubkey-39db7c82-5f68629b.asc
- ▶ gpg-pubkey-50a3dd1c-50f35137.asc
- ▶ ls-IR.gz
- ▶ media.1
- ▶ repodata
- ▶ suse.ins
- ▶ suse_ptf_key.asc
- ▶ susehmc.ins

4.3 Collecting information

Based on the instructions that are provided in the planning stage as described in Chapter 2, “Planning for the Kernel-based Virtual Machine host and guest” on page 21, it is recommended that you save the information that you use during the installation process.

A good practice is to create a table (see Table 4-2) that contains the components information. This table is useful during the installation process.

Table 4-2 Sample KVM host installation checklist

KVM host installation checklist			
Name	Type	Description	More information
Host IP/subnet	TCP/IP	129.40.23.197/24	KVM host
	VLAN	8	
Hostname.domain	DNS	rdbkkvm3.pbm.ihost.com	DNS server 129.40.106.1
Gateway	Default GW	129.40.22.254	
FTP server	FTP port 20/21	rdbkftp1.pbm.ihost.com	Check firewall rules
FTP folder	Install folder	/SLES15SP3IDVD1	Check permission
FTP access	Credentials	user: Inxadmin pw xxx	
LPAR	Logical Partition	Aries37	
Memory	RAIM Memory	32 GB	HostOS, GuestsOS, and Workloads
Physical Processors	IFL (shared)	16 IFL	SMT enabled
Virtual Processors	Virtual Processors	Two for each guest	Can be expanded later, recommended vCPU number <=max of physical CPUs
Storage	ECKD DASD	0.0.90DE 0.0.904B 0.0.914B	400 GB 54 GB 54 GB
OSA1	Network card1	CHP E8	Devices 1E80-1E82
CRYPTO	Domain/Card	CARDS 0x03 0x06	DOMAINS 0x1E 0x1F 0x20

4.3.1 Required information for SLES on an LPAR installation

In this section, we describe our lab environment. You can use the information in this section as a reference to create your own environment.

Installation by using FTP

SLES can be installed from a DVD in the HMC or from an FTP server. In this example, we installed SLES from an FTP server. Be sure to have the FTP port open in the firewall. Our lab environment included the following FTP server information:

- ▶ IP address: rdbkftp1.pbm.ihost.com
- ▶ Credentials:
 - User lnxadmin
 - Password xxxxxx
- ▶ Directory: /SLES15SP3IDVD1

OSA device addresses

On the IBM Z platform, the network interface cards (NIC) are represented by OSA Express adapters. Each OSA card can manage a range of devices. To use a specific OSA, three consecutive addresses are required: one device for control reads, one for control writes, and the third for data.

For this example, we choose the first triplet from OSA CHPID E2 (1E80-1E82).

Networking information

Contact your network administrator for the correct networking information for the host.

The following networking information was used in our lab environment:

- ▶ Hostname: rdbkkvm3
- ▶ IP address: 129.40.23.197
- ▶ Subnet prefix: 24
- ▶ Default gateway: 129.40.23.254
- ▶ Layer 2 or 3: 2
- ▶ VLAN: 8
- ▶ DNS: 129.40.106.1 and 129.40.106.2

IP address 100.150.233.60 was used for HiperSockets network access.

Storage

As described 2.2.2, “Storage considerations” on page 27, two options are available for storage on the Linux on IBM Z platform: ECKD DASD disk or FCP LUN disk.

In this example, we used ECKD DASD.

Our storage included the following information:

- ▶ ECKD device address: 90de
- ▶ Volume serial: 0X90DE
- ▶ Space: 400

The operating system installation uses a single DASD under Logical Volume Manager (LVM).

4.3.2 Required information for VM installations

In this section, we review the following required information for VM installations:

- ▶ Compute
- ▶ Memory
- ▶ Disk
- ▶ Network
- ▶ Cryptography

Compute

For VM deployment, all of the guests use two virtual CPUs (vCPU) to use the Simultaneous Multi-Threading (SMT) on an IBM Integrated Facility for Linux (IFL) processor.

Memory

Each VM has 2 GB of RAM, which is the amount of memory that is related to the type of workload that a machine is going to host. For the Linux guest operating system, we recommend starting with 512 MB of memory (for more information, see Chapter 2, “Planning for the Kernel-based Virtual Machine host and guest” on page 21).

To avoid memory constraints, it is a good practice to have an accurate workload and capacity study to suitably define the amount of memory.

Disk

QEMU Copy On Write (QCOW2) is a file format for disk image files that are used by Quick Emulator (QEMU), which is a hosted VM monitor. QCOW2 uses a disk storage optimization strategy that delays allocation of storage until it is needed. Disk images for specific guest operating systems often are provided as a file in the QCOW2 format.

A QCOW2 image file was used for the operating system disk in our example.

The files were stored in the LVM to create more flexible storage migrations. For more information, see 2.2.2, “Storage considerations” on page 27.

The ECKD-DASD used for the Volume Group (VG) that is used for images (rdbkkvm3-images), is the 0X904B volume.

The maximum space that is specified in our lab environment for the image files was 10 GB, although this maximum can be extended.

We created the following two disk images to use as storage for the VM guests:

- ▶ kvm3guest01: /var/lib/libvirt/images/kvm3guest1_vol001.img
- ▶ kvm3guest02: /var/lib/libvirt/images/kvm3guest2_vol001.img
- ▶ kvm3guest03: /var/lib/libvirt/images/kvm3guest3_vol001.img
- ▶ kvm3guest04: /var/lib/libvirt/images/kvm3guest4_vol001.img

Network

As described in “OSA device addresses” on page 103, contact your network team for the networking information.

Our lab environment used the following network setup:

- ▶ Hostname: kvm3guest01
- ▶ IP address: 129.40.23.208

- ▶ Subnet prefix: 24
- ▶ Default gateway: 129.40.23.254
- ▶ Hostname: kvm3guest02
- ▶ IP address: 129.40.23.209
- ▶ Subnet prefix: 24
- ▶ Default gateway: 129.40.23.254
- ▶ Hostname: kvm3guest03
- ▶ IP address: 129.40.23.210
- ▶ Subnet prefix: 24
- ▶ Default gateway: 129.40.23.254
- ▶ Hostname: kvm3guest04
- ▶ IP address: 129.40.23.211
- ▶ Subnet prefix: 24
- ▶ Default gateway: 129.40.23.254
- ▶ For HiperSockets access:
 - Hostname: rdbkkvm3
 - IP address: 100.150.233.60
 - Hostname: kvm3guest01
 - IP address: 100.150.233.61
 - Hostname: kvm3guest02
 - IP address: 100.150.233.62
 - Hostname: kvm3guest03
 - IP address: 100.150.233.63
 - Hostname: kvm3guest04
 - IP address: 100.150.233.64

Cryptography

For more information about the z15 Crypto Express adapters, see 2.4.5, “Cryptography” on page 42.

In our lab environment, we assigned four crypto adapters and three domains to ARIES37 LPAR.

The Adjunct Processor (AP) queues that we used in our lab environment as our virtual cryptographic resources are listed in Table 4-3.

Table 4-3 AP queues assignment

Crypto domains/Crypto adapters	03 (0x03)	06 (0x6)
30 (0x1E)	03.001E	06.001E
31 (0x4c)	03.001F	06.001F
32 (0x20)	03.0020	06.0020

As described in 2.4.5, “Cryptography” on page 42, the AP queues are a combination of <crypto card>.<crypto domain>., both expressed in hexadecimal. Consider the following points:

- ▶ Domain 30 was used for rdbkkvm3
- ▶ Domain 31 was used for kvm3guest1
- ▶ Domain 32 was used for kvm3guest2

4.4 Installing SUSE on an LPAR as a KVM host

In this section, we describe how to perform the following tasks:

- ▶ Prepare for the installation
- ▶ Install SLES on an LPAR
- ▶ Prepare the host for virtualization

4.4.1 Preparing the installation

The information that we created to use an FTP server to install SLES on an LPAR is described in “Installation by using FTP” on page 103.

In this example, we created a directory structure that contained the .ins and .p files that are needed for the installer for SLES on an LPAR.

Example 4-1 shows the contents of the rdbkkvm3.ins file, which is a copy of the suse.ins file that is provided in the root of the SLES ISO installer. Change only the line boot/s390x/parmfile, replacing parmfile with rdbkkvms.p.

Example 4-1 rdbkkvm3.ins

```
* SUSE Linux for IBM Z Installation/Rescue System
boot/s390x/linux 0x00000000
boot/s390x/initrd.off 0x0001040c
boot/s390x/initrd.siz 0x00010414
boot/s390x/initrd 0x01000000
boot/s390x/rdbkkvm3.p 0x00010480
```

Example 4-2 shows the rdbkkvms.p file. It defines DASD for the target installation (or the SCSI information if you have FCP SAN disk), network properties, and the location of the FTP repository.

Example 4-2 rdbkkvm3.p parameter file

```
ro ramdisk_size=50000 PORTNO=0 InstNetDev=osa OSAInterface=qdio osahwaddr=
cio_ignore=all,!condev,!0.0.1e80-0.0.1e82,!0.0.90de
ReadChannel=0.0.1e80 WriteChannel=0.0.1e81 DataChannel=0.0.1e82
HostIP=129.40.23.197/24 Gateway=129.40.23.254 Hostname=rdbkkvm3
Domain=pbm.ihost.com nameserver=129.40.106.1 vlanid=8
Install=ftp://129.40.23.88/SLES15SP3IDVD1
usevnc=1 vncpassword=lnx4rdbk ssl_certs=0
self_update=0
```

Consider the following points:

- ▶ The `rd.dasd` statement points to our storage ECKD disk device.
- ▶ The `cio_ignore=all` disables all devices and with the `!condev`, allows you to specify which devices are to be available at the installation time.

Note: When the `cio_ignore=all` is used to add new devices, you must use the `chzdev` command to update the kernel enabled devices.

- ▶ The following parameters are related to networking:
 - `readchannel`, `writechannel`, and `datachannel`, which are the devices of the OSA triplet.
 - IP parameters: `hostname`, `hostip`, `netmask`, `broadcast`, and `gateway`.
- ▶ The `install` parameter points to the source of the installation DVD1.
- ▶ The following console parameters were used:
 - `linuxrclog`, which is related to console
 - VNC parameters: `vnc=1` (VNC enabled) and `password VNCPassword`

4.4.2 Installing SLES on an LPAR

After all of the prerequisites were met, we started from FTP by using the information that is described in “Installation by using FTP” on page 103 (see Figure 4-4).

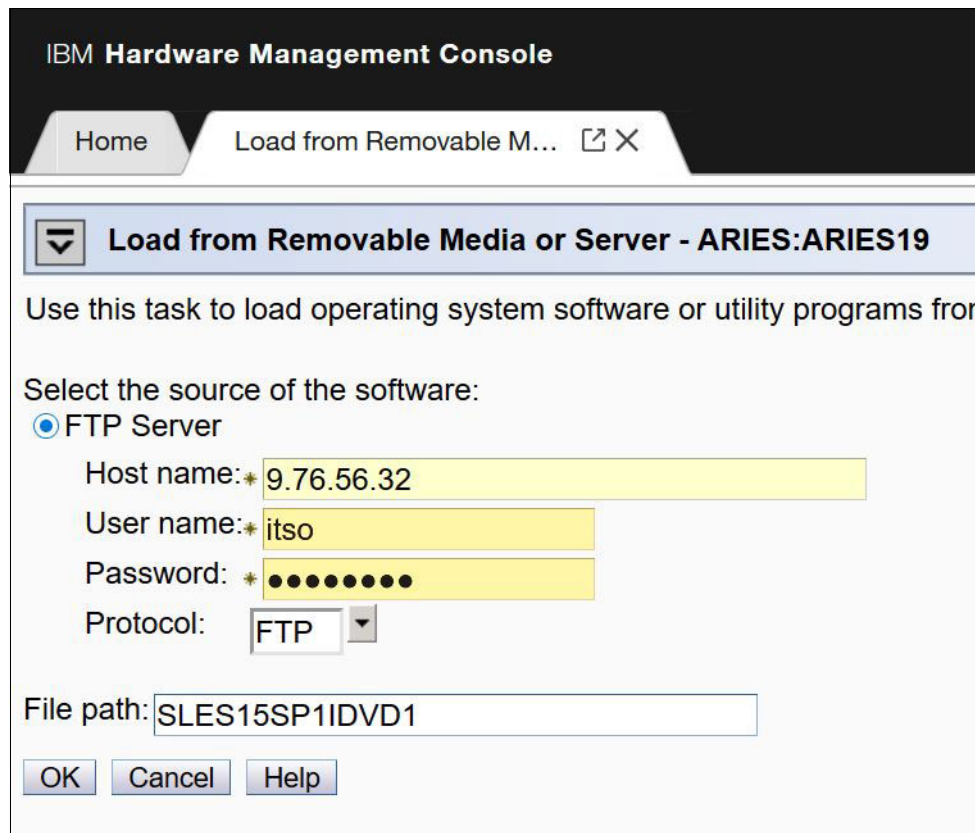


Figure 4-4 Loading from an FTP server

When you receive the prompt with the list of .ins files in the DPM or HMC, choose the file that you created, such as rdbkvm3.ins.

Continue with the installation process. For more information, this [SUSE web page](#).

4.5 Preparing the host for virtualization

Complete the following steps to enable SLES on IBM Z as a KVM Host:

1. Subscribe the server to the SUSE Repository Mirroring Tool (RMT).

To access the packages and support, you must subscribe your system to a SUSE RMT server. For more information about this process, see SUSE's [Repository Mirroring Tool Guide](#).

Also, you can install and update packages from a local repository. For more information, see this [SUSE web page](#).

2. Check whether the LPAR supports virtualization functions.

The LPAR must support Start Interpretive Execution (SIE) instructions, as shown in Example 4-3.

Example 4-3 Checking virtualization support

```
rdbkvm3:/home/lnxadmin # lscpu | grep sie
Flags:                esan3 zarch stfle msa ldisp eimm dfp edat etf3eh
highprsr te vx vxd vxe gs vxe2 vxp sort dflt sie
```

3. Load the KVM module and verify that the loading process was successful.

As shown in Example 4-4, issue the Linux command to load the KVM module by using the modprobe command, and validate that KVM is loaded by using the command **lsmod**.

Example 4-4 Loading KVM module

```
rdbkvm3:/home/lnxadmin # modprobe kvm
rdbkvm3:/home/lnxadmin # lsmod | grep kvm
kvm                417792  0
```

4. Install the virtualization packages and modules.

It is important to install the virtualization module during the LPAR installation (as shown in Figure 4-5) by choosing the **KVM Virtualization Host and tools** and **KVM Host Server** option during the SLES installation process.

Note: The figures that are presented here show a graphical installation. To use this method, it is necessary to configure VNC in the guest. For more information about installing and configuring VNC, see this [SUSE Documentation web page](#).

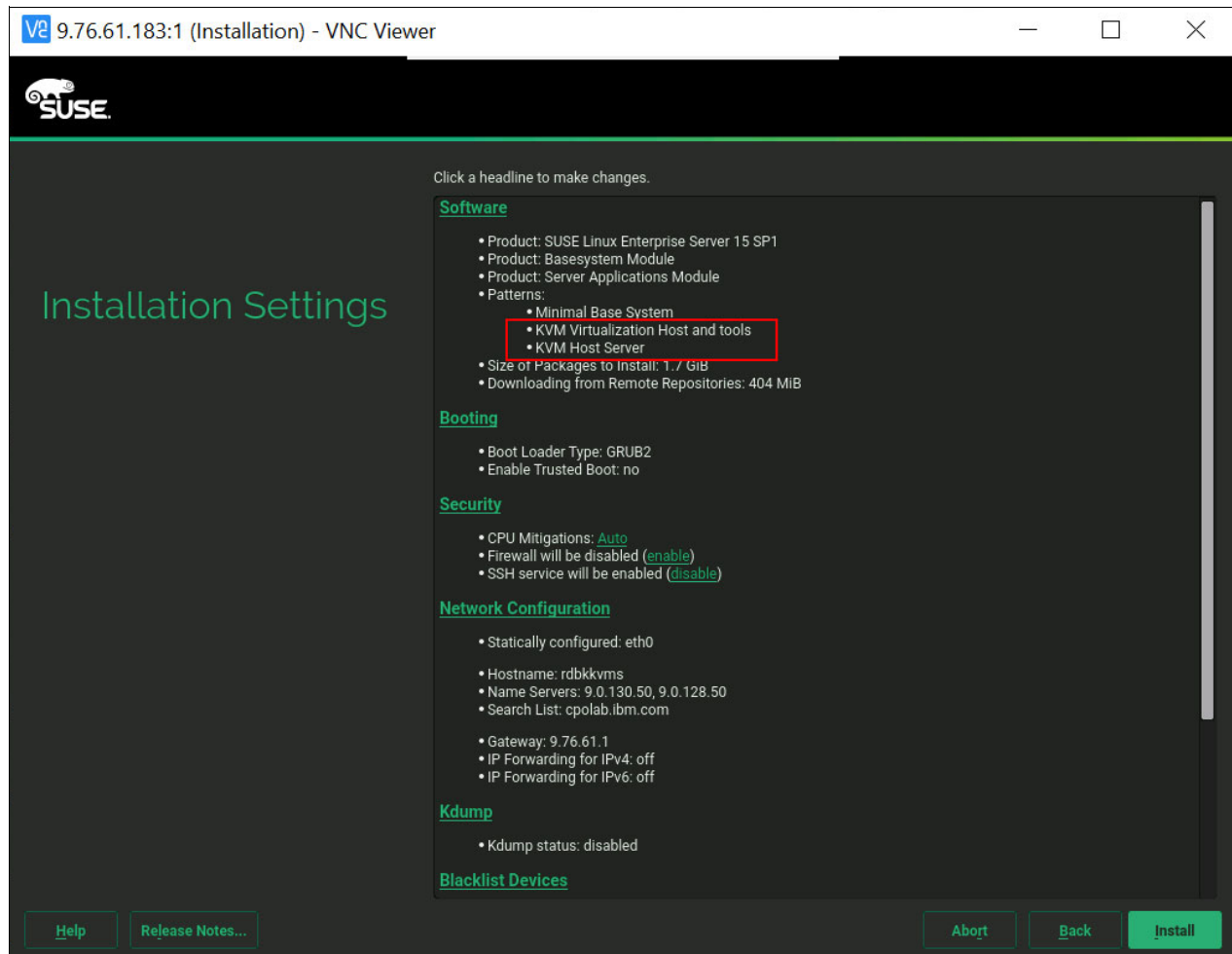


Figure 4-5 SUSE software selection during installation

Alternatively, you can install the virtualization packages later by running the command that is shown in Example 4-5.

Example 4-5 Installing KVM packages

```
rdbkvm3:/home/lnxadmin # zypper install virt-manager virt-viewer qemu kvm libvirt  
libvirt-python virt-install
```

5. Validate whether the host is ready for virtualization.

Before starting to work with KVM, run the **virt-host-validate** command, as shown in Example 4-6.

Example 4-6 Virtualization verification

```
rdbkvm3:/home/lnxadmin # virt-host-validate
QEMU: Checking for hardware virtualization :
PASS
QEMU: Checking if device /dev/kvm exists :
PASS
QEMU: Checking if device /dev/kvm is accessible :
PASS
QEMU: Checking if device /dev/vhost-net exists :
PASS
QEMU: Checking if device /dev/net/tun exists :
PASS
QEMU: Checking for cgroup 'cpu' controller support :
PASS
QEMU: Checking for cgroup 'cpuacct' controller support :
PASS
QEMU: Checking for cgroup 'cpuset' controller support :
PASS
QEMU: Checking for cgroup 'memory' controller support :
PASS
QEMU: Checking for cgroup 'devices' controller support :
PASS
QEMU: Checking for cgroup 'blkio' controller support :
PASS
QEMU: Checking if IOMMU is enabled by kernel :
PASS
QEMU: Checking for secure guest support :
WARN (IBM Secure Execution appears to be disabled in kernel. Add prot_virt=1 to
kernel cmdline arguments)
LXC: Checking for Linux >= 2.6.26
: PASS
LXC: Checking for namespace ipc :
PASS
LXC: Checking for namespace mnt :
PASS
LXC: Checking for namespace pid :
PASS
LXC: Checking for namespace uts :
PASS
LXC: Checking for namespace net :
PASS
LXC: Checking for namespace user :
PASS
LXC: Checking for cgroup 'cpu' controller support :
PASS
LXC: Checking for cgroup 'cpuacct' controller support :
PASS
LXC: Checking for cgroup 'cpuset' controller support :
PASS
LXC: Checking for cgroup 'memory' controller support :
PASS
```

```

LXC: Checking for cgroup 'devices' controller support      :
PASS
LXC: Checking for cgroup 'freezer' controller support     :
PASS
LXC: Checking for cgroup 'blkio' controller support       :
PASS
LXC: Checking if device /sys/fs/fuse/connections exists   :
PASS

```

When KVM is used, you must check only the QEMU tests, as shown in Example 4-6 on page 110. The LXC test results are for Linux containers.

The WARN in the last line indicates that this host is not enabled to use secure guest support.

For more information about secure guest support on IBM z15 and how to secure this feature, see Chapter 8, “Using IBM Secure Execution” on page 253.

For more information about planning and implementation, see this [IBM Documentation web page](#).

4.6 Configuring the KVM host

This section describes how to enable SLES as a KVM host and set up the devices to be ready for VM guest usage.

4.6.1 Defining NICs

As described in 4.1, “Defining the target configuration” on page 96, we use in our lab environment one NIC through the 1e80-1e82 triplet OSA devices (which is defined in the E8 OSA channel) for management purposes.

For the VM guest network, we used the MacVTap network that uses a bond interface with two OSA interfaces (OSA E8 and OSA EE).

As shown in Example 4-7, the only NIC that is configured is the NIC that we used for the SLES installation.

Example 4-7 Configured networks

```

rdbkvm3:~ # znetconf -c

```

Device IDs	Type	Card Type	CHPID	Drv. Name	State
0.0.1e80,0.0.1e81,0.0.1e82	1731/01	OSD_10GIG	E8	qeth eth0	online

By following the architecture that is used in our lab environment for the guest network, we must add two NICs (OSA triplets) that use different OSA cards that access the same network through different switches.

Example 4-8 shows two unconfigured NICs that were added with different OSA cards and CHPIDs, which provide redundancy for the virtual environment.

Example 4-8 Checking NICS availability

```
rdbkkvm3:~ # znetconf -u
Scanning for network devices...
Device IDs          Type      Card Type      CHPID Drv.
-----
0.0.1e83,0.0.1e84,0.0.1e85 1731/01 OSA (QDIO)      e8 qeth
0.0.1ee3,0.0.1ee4,0.0.1ee5 1731/01 OSA (QDIO)      ee qeth
```

As shown in Example 4-9, we configured the 0.0.1e83-0.0.1e85 device as interface eth5 and the 0.0.1ee3-0-0.0.0.1ee5 device as interface eth6.

Example 4-9 Configuring the NICs

```
rdbkkvm3:/etc/sysconfig/network # chzdev -e qeth 0.0.1e83,0.0.1e84,0.0.1e85 layer2=1
buffer_count=128
QETH device 0.0.1e83:0.0.1e84:0.0.1e85 configured
rdbkkvm3:/etc/sysconfig/network # chzdev -e qeth 0.0.1ee3,0.0.1ee4,0.0.1ee5 layer2=1
buffer_count=128
QETH device 0.0.1ee3:0.0.1ee4:0.0.1ee5 configured
```

Example 4-10 shows how to validate interfaces eth5 and eth6.

Example 4-10 Validating interfaces

```
rdbkkvm3:~ # ip link | grep 'eth5\|eth6'
8: eth5: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group default
qlen 1000
9: eth6: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group default
qlen 1000
```

For more information about network configuration on SLES, see this [IBM Documentation web page](#).

4.6.2 Defining the bond interface

To enable network high availability (HA), we define a bond interface that is named bond0 (master). This interface accesses the physical network through two NIC slave interfaces: eth5 and eth6.

Example 4-11 shows how to define a bond interface and set eth5 and eth6 as slave interfaces of the bond0 interface. To allow guest traffic through the bond interface, the slave interfaces must be in promiscuous mode. Promiscuous mode allows a network device to intercept and read each network packet; therefore, these interfaces must be disabled to change the properties of NICs.

Example 4-11 Defining a bond interface

```
rdbkkvm3:~ # ip link add bond0 type bond miimon 100 mode balance-tlb
rdbkkvm3:~ # ip link set eth5 down
rdbkkvm3:~ # ip link set eth6 down
rdbkkvm3:~ # ip link set eth5 master bond0
rdbkkvm3:~ # ip link set eth6 master bond0
rdbkkvm3:~ # ip link set eth5 up
```



```
rdbkkvm3:~ # ip link set eth6 up
rdbkkvm3:~ # ip link set bond0 up
```

If dedicated OSAs are used for the slaves and the proper configuration is used for the switches, you can configure the bonding option as `mode=802.3ad miimon=100`, which allows you to create LACP aggregation groups that share the speed and duplex settings.

If two OSA Express7s 10 Gb are used, you can aggregate two 10 Gb per second (Gbps) ports into a 20 Gbps trunk port. This aggregation is equivalent to having one interface with 20 Gbps speed. It provides fault tolerance and load balancing.

As shown in Example 4-12, we verify that the definition of the `bond0` interface is correct.

Example 4-12 Verifying bond interface

```
rdbkkvm3:~ # cat /proc/net/bonding/bond0
Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)
```

```
Bonding Mode: transmit load balancing
Primary Slave: None
Currently Active Slave: eth5
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0
Peer Notification Delay (ms): 0
```

```
Slave Interface: eth5
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 36:7a:07:65:28:ce
Slave queue ID: 0
```

```
Slave Interface: eth6
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: ee:a1:0b:5e:1d:b6
Slave queue ID: 0
```

Next, we must set the `bond0` interface and the slave configuration as permanent in the `eth*` interfaces.

Example 4-13 shows the content of our definition file for the channel bonding interface, `bond0`.

Example 4-13 Making bondn interface permanent-bonding master configuration file

```
rdbkkvm3:/etc/sysconfig/network # cat ifcfg-bond0
IPADDR='0.0.0.0'
BOOTPROTO='none'
STARTMODE='auto'
BONDING_MASTER='yes'
BONDING_SLAVE0='eth5'
BONDING_SLAVE1='eth6'
BONDING_MODULE_OPTS='mode=balance-tlb miimon=100'
```

Example 4-14 shows how to make the VLAN interface configuration.

Example 4-14 Making a VLAN interface bond0.8 interface permanent configuration file

```
rdbkkvm3:/etc/sysconfig/network # cat ifcfg-bond0.8
IPADDR='0.0.0.0/32'
NAME=''
MTU='0'
BOOTPROTO='static'
STARTMODE='auto'
ZONE=''
ETHERDEVICE='bond0'
VLAN_ID='8'
```

Example shows how to code the permanent slave configuration file.

Example 4-15 Coding the eth6 interface permanent slave configuration file

```
IPADDR='0.0.0.0'
NAME=''
BOOTPROTO='none'
STARTMODE='auto'
ZONE=''
rdbkkvm3:/etc/sysconfig/network # cat ifcfg-eth6
IPADDR='0.0.0.0'
NAME=''
BOOTPROTO='none'
STARTMODE='auto'
ZONE=''
```

For more information about bonding, see this [IBM Documentation web page](#).

Also, we can configure networking parameters to SLES configurations by using `yast`, which is a user-friendly interface. For more information about configuring bond interfaces on `yast`, see this [SUSE Documentation web page](#).

4.6.3 Defining HiperSockets interfaces

HiperSockets allows memory-to-memory communication between hosts in the same IBM Z platform. HiperSockets avoids the use of external communications by way of an NIC and Ethernet switch, which eliminates traditional network latency. For more information about this feature, see “Network connectivity” on page 6.

As described in 4.1, “Defining the target configuration” on page 96, the HiperSocket CHPID is F4 in our lab environment, and triplet for the `hsi0` interface definition is 0F00-0F02.

4.6.4 Defining the HiperSocket interface to support VM guest network

We define the encf00 interface on the HiperSocket chipid(F4) to allow VM guest access to the HiperSocket network.

Example 4-16 on page 115 shows the HiperSocket device availability.

Example 4-16 List of unconfigured HSI devices on F4 CHPID

```
rdbkkvm3:/home/lnxadmin # znetconf -u
Scanning for network devices...
Device IDs          Type      Card Type      CHPID Drv.
-----
0.0.0f00,0.0.0f01,0.0.0f02 1731/05 HiperSockets   f4 qeth
0.0.0f03,0.0.0f04,0.0.0f05 1731/05 HiperSockets   f4 qeth
```

Choose the 0.0.0f00,0.0.0f01,0.0.0f02 devices to create the hsi0 interface, as shown in Example 4-17.

Example 4-17 Configuring the HiperSocket interface

```
rdbkkvm3:/home/lnxadmin # chzdev -e qeth 0.0.0f00,0.0.0f01,0.0.0f02 layer2=1
buffer_count=128
QETH device 0.0.0f00:0.0.0f01:0.0.0f02 configured
```

Next, you validate the new interface, as shown in Example 4-18.

Example 4-18 Validating HiperSockets interface

```
rdbkkvm3:/home/lnxadmin # ip link show hsi0
12: hsi0: <BROADCAST,MULTICAST> mtu 8192 qdisc noop state DOWN mode DEFAULT group
default qlen 1000
    link/ether 0e:00:f4:37:00:02 brd ff:ff:ff:ff:ff:ff
    altname encf00
```

4.6.5 Defining the HiperSocket interface of the KVM host

We need to define a HiperSockets interface for KVM use. For this definition, select the 0.0.0f03,0.0.0f04,0.0.0f05 devices to create interface hsi1, as shown in Example 4-19.

Example 4-19 Configuring the HiperSocket interface

```
rdbkkvm3:/home/lnxadmin # chzdev -e qeth 0.0.0f03,0.0.0f04,0.0.0f05 layer2=1
buffer_count=128
QETH device 0.0.0f03:0.0.0f04:0.0.0f05 configured
```

Assign the IP address to the interface and start the interface, as shown in Example 4-20.

Example 4-20 Assigning IP address and start HSI1 interface

```
rdbkkvm3:/home/lnxadmin # ip addr add 100.150.233.60/24 dev hsi1
rdbkkvm3:/home/lnxadmin # ip link set hsi1 up
```

Example 4-21 shows the contents of the file, `ifcfg-hsi1`, which is found in the `/etc/sysconfig/network` directory to have persistency for the HSI1 interface.

Example 4-21 Making hsi1 interface configuration permanent

```

rdbkvm3:/etc/sysconfig/network # cat ifcfg-hsi0
IPADDR='0.0.0.0/32'
NAME=''
BOOTPROTO='static'
STARTMODE='auto'
ZONE=''
rdbkvm3:/etc/sysconfig/network # cat ifcfg-hsi1
IPADDR='100.150.233.60/24'
NAME=''
BOOTPROTO='static'
STARTMODE='auto'
ZONE=''

```

4.6.6 Defining HiperSocket Converged Interface

By using HiperSockets Converged Interface (HSCI) connections, a HiperSockets network interface can be combined with an external OSA- or RoCE port, which creates a single network interface. With this interface, we can access the switched network and the intra-CEC HiperSocket network with the same IP. Both of the devices that participate in the HSCI interface must have the same physical network (PNET) ID.

For our lab, we choose the adapters that are listed in Table 4-4.

Table 4-4 Adapters that were used in our lab environment

Device Type	CHPID	Devices	PNETID
HiperSocket	F2	0.0.0FC9 0.0.0FCA 0.0.0FCB	PERFNET
OSA Express	EE	0.0.1EE9 0.0.1EEA 0.0.1EEB	PERFNET

Example 4-22 - Example 4-27 on page 117 show the steps that are required to define the HiperSocket Converged Interface.

Example 4-22 Checking PNETID of the HiperSocket device

```

rdbkvm3:/home/lxadmin/isos # cat /sys/devices/css0/chp0.f2/util_string | iconv
-f IBM-1047 -t ASCII
PERFNET

```

Example 4-23 Checking the OSA PNETID

```

rdbkvm3:/home/lxadmin/isos # cat /sys/devices/css0/chp0.ee/util_string | iconv
-f IBM-1047 -t ASCII
PERFNET

```

Example 4-24 Creating HSI and OSA interfaces

```
rdbkvm3:/home/lxadmin/isos # chzdev -e qeth 0.0.0fc9,0.0.0fca,0.0.0fcb layer2=1
buffer_count=12
QETH device 0.0.0fc9:0.0.0fca:0.0.0fcb configured
rdbkvm3:/home/lxadmin/isos # chzdev -e qeth 0.0.1ee9,0.0.1eea,0.0.1eeb layer2=1
buffer_count=12
QETH device 0.0.1ee9:0.0.1eea:0.0.1eeb configured
```

Example 4-25 Checking the HSCI interface

```
rdbkvm3:/home/lxadmin/isos # hsci add hsi2 eth10
Verifying net dev eth10 and HiperSockets dev hsi2
Adding hsci0fc9 with a HiperSockets dev hsi2 and an external dev eth10
Set hsi2 MAC 0e:00:f2:37:00:0b on eth10 and hsci0fc9
Successfully added HSCI interface hsci0fc9
```

Example 4-26 Creating the VLAN 8 interface from hsci0fc9 device and assign IP

```
rdbkvm3:/home/lxadmin/isos # ip link add dev hsci0fc9.8 link hsci0fc9 type vlan
id 8
rdbkvm3:/home/lxadmin/isos # ip addr add 129.40.23.230/24 dev hsci0fc9.8
rdbkvm3:/home/lxadmin/isos # ip link set up hsci0fc9.8
```

Example 4-27 Checking HSCI interface

```
rdbkvm3:/home/lxadmin/isos # hsci show
HSCI      PNET_ID      HiperSockets      External
-----
hsci0fc9  PERFNET      hsi2               eth10
```

4.6.7 Defining SMC interfaces

SMC-R and SMC-D use shared memory to provide low-latency, high-bandwidth, cross-LPAR connections for applications. This support is intended to provide application-transparent direct memory access (DMA) communications to TCP endpoints for socket-based connections.

SMC tools package installation

To support SMC-D (ISM) and SMC-R (RoCE), you must install the `smc-tools` package. For more information about obtaining the packages, see this [GitHub web page](#). After the content is downloaded, decompress the packages and upload them to the host (in our example, to `/home/isos/`).

Use the commands that are shown in Example 4-28 to install the packages.

Example 4-28 Installing SMC tools package

```
rdbkvm3:/home/lxadmin/isos # zypper install smc-tools
Loading repository data...
Reading installed packages...
Resolving package dependencies...
```

The following NEW package is going to be installed:
smc-tools

```

1 new package to install.
Overall download size: 62.0 KiB. Already cached: 0 B. After the operation,
additional 161.4 KiB will be used.
Continue? [y/n/v/...? shows all options] (y): y
Retrieving package smc-tools-1.5.0-1.8.s390x
(1/1), 62.0 KiB (161.4 KiB unpacked)
Retrieving: smc-tools-1.5.0-1.8.s390x.rpm
.....
.....[done]

Checking for file conflicts:
.....
.....[done]
(1/1) Installing: smc-tools-1.5.0-1.8.s390x
.....
.....[done]
rdbkkvms:/var/lib/libvirt/images # cd smc-tools-1.2.0
rdbkkvms:/var/lib/libvirt/images/smc-tools-1.2.0 # zypper install libn*
rdbkkvms:/var/lib/libvirt/images/smc-tools-1.2.0 # make

```

SMC-D

In this section, we provide the basic commands to enable SMC-D on the SLES host server.

Example 4-29 shows how to check the ISM device availability.

Example 4-29 Checking PCI devices

```

rdbkkvm3:/home/lxadmin/isos # smc_rnics

```

FID	Power	PCI_ID	PCHID	Type	PPrt	PNET_ID	Net-Dev
c7	1	0005:00:00.0	0158	RoCE_Express2	n/a	PERFNET	eth7
d7	1	0006:00:00.0	0158	RoCE_Express2	n/a	PERFNET	eth8
e7	1	0003:00:00.0	01b8	RoCE_Express2	n/a	PERFNET	eth1
f7	1	0004:00:00.0	01b8	RoCE_Express2	n/a	PERFNET	eth2
1036	1	0000:00:00.0	07c0	ISM	n/a	PERFNET	n/a
1136	1	0001:00:00.0	07c1	ISM	n/a	PERFNET1	n/a
1236	1	0002:00:00.0	07c2	ISM	n/a	PERFNET2	n/a

As shown in Example 4-30 and Example 4-31, we check the PNET ID of the ISM device and in the OSA. Both should display the same PNET ID.

Example 4-30 Checking PNET ID of the ISM device

```
rdbkkvm3:/home/lnxadmin/isos # cat
/sys/devices/pci0000:00/0000:00:00.0/util_string | iconv -f IBM-1047 -t ASCII
PERFNET
```

Example 4-31 Checking the OSA PNET ID

```
rdbkkvm3:/home/lnxadmin/isos # cat /sys/devices/css0/chp0.ee/util_string | iconv
-f IBM-1047 -t ASCII
PERFNET
```

In our lab, we define an NIC in CHPID EE (see 4.6.1, “Defining NICs” on page 111) with the command that is shown in Example 4-32.

Example 4-32 Defining OSA and assign IP

```
rdbkkvm3:/home/lnxadmin/isos # chzdev -e qeth 0.0.1ee6,0.0.1ee7,0.0.1ee8 layer2=1
buffer_count=128
QETH device 0.0.1ee6:0.0.1ee7:0.0.1ee8 configured
```

Note: Assign IP address 129.40.23.223/24 through **Yast**. If a VLAN is used, set the IP on the VLAN interface.

To test the communication between rdbkkvm3 and rdbkkvm2 LPARs in the same CPC that use the SMC-D, we use the iperf3 tool. To install it, run the yum command that is shown in Example 4-33 in each LPAR.

Example 4-33 Installing iperf

```
rdbkkvm3:/home/lnxadmin/isos # zypper addrepo
https://download.opensuse.org/repositories/network:utilities/SLE_15_SP3/network:ut
ilities.repo
rdbkkvm3:/home/lnxadmin/isos # zypper refresh
*respond "t" for allowing temporary access or "a" always
rdbkkvm3:/home/lnxadmin/isos # zypper install iperf
[root@rdbkkvm2 home]# yum -y install iperf3
```

For more information, see this [iopenSUSE.org web page](https://www.iopensuse.org).

Allow the local firewall to accept connections for iperf3 on the 5201 TCP port on rdbkkvm2 LPAR server (see Example 4-43).

Example 4-34 Allowing firewall port 5201

```
[root@rdbkkvm2 home]# firewall-cmd --permanent --add-port=5201/tcp
success
[root@rdbkkvm2 home]# firewall-cmd --reload
success
[root@rdbkkvm2 ~]# firewall-cmd --list-all
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: bond1 bond1.008 enP6p0s0 enP6p0s0.008 enc1e80 enc1e80.008 enc1e83
enclee3 enclee6 enclee6.008 encf00 encf03
  sources:
  services: cockpit dhcpv6-client ssh
  ports: 21/tcp 5901/tcp 5201/tcp
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

Start iperf3 in listening mode by using the command that is shown in Example 4-35 on rdbkkvm2.

Example 4-35 Starting iperf3 by using SMC

```
[root@rdbkkvm2 ~]# smc_run iperf3 -s
-----
Server listening on 5201
-----
```

Use the command that is shown in Example 4-36 to open another SSH session against the rdbkkvm2 server and print the information about the SMC sockets.

Example 4-36 Checking the SMC listening on port 5201

```
[root@rdbkkvm2 ~]# smcss -a
State          UID  Inode  Local Address          Peer Address          Intf
Mode
LISTEN         00000 0526700 0.0.0.0:5201
```

To test SMC connections, issue the iperf3 command that is shown in Example 4-37 on rdbkkvm1 and check on rdbkkvm2 (see Example 4-37 and Example 4-38 on page 121).

Example 4-37 Running iperf3 client on rdbkkvm3 to the server on rdbkkvm2

```
rdbkkvm3:/home/lnxadmin/isos # smc_run iperf3 -c 129.40.23.221 -t 10
Connecting to host 129.40.23.221, port 5201
[ 5] local 129.40.23.197 port 40958 connected to 129.40.23.221 port 5201
[ ID] Interval          Transfer          Bitrate          Retr  Cwnd
[ 5]  0.00-1.00    sec  2.86 GBytes     24.6 Gbits/sec    0  14.1 KBytes
[ 5]  1.00-2.00    sec  2.88 GBytes     24.8 Gbits/sec    0  14.1 KBytes
[ 5]  2.00-3.00    sec  2.91 GBytes     25.0 Gbits/sec    0  14.1 KBytes
```



```

[ 5] 3.00-4.00 sec 2.81 GBytes 24.1 Gbits/sec 0 14.1 KBytes
[ 5] 4.00-5.00 sec 2.80 GBytes 24.0 Gbits/sec 0 14.1 KBytes
[ 5] 5.00-6.00 sec 2.72 GBytes 23.4 Gbits/sec 0 14.1 KBytes
[ 5] 6.00-7.00 sec 2.68 GBytes 23.0 Gbits/sec 0 14.1 KBytes
[ 5] 7.00-8.00 sec 2.71 GBytes 23.3 Gbits/sec 0 14.1 KBytes
[ 5] 8.00-9.00 sec 2.70 GBytes 23.2 Gbits/sec 0 14.1 KBytes
[ 5] 9.00-10.00 sec 2.75 GBytes 23.6 Gbits/sec 0 14.1 KBytes
-----
[ ID] Interval          Transfer      Bitrate      Retr
[ 5] 0.00-10.00 sec 27.8 GBytes 23.9 Gbits/sec 0          sender
[ 5] 0.00-10.00 sec 27.8 GBytes 23.9 Gbits/sec          receiver

```

iperf Done.

Example 4-38 Checking the usage of SMC-D

```

[root@rdbkvm2 ~]# smcss -a
State      UID      Inode    Local Address          Peer Address          Intf
Mode
ACTIVE     00000   0129536  ::ffff:129.40.23...:5201 ::ffff:129.40.2...:40950 0000
SMCD
ACTIVE     00000   0129535  ::ffff:129.40.23...:5201 ::ffff:129.40.2...:40948 0000
SMCD
LISTEN     00000   0129522  0.0.0.0:5201
[root@rdbkvm2 ~]# smcss -D
State      UID      Inode    Local Address          Peer Address          Intf
Mode GID          Token          Peer-GID          Peer-Token          Linkid
ACTIVE     00000   0129536  ::ffff:129.40.23...:5201 ::ffff:129.40.2...:40950 0000
SMCD 11000facb7f88561 0000090810000000 25000fadb7f88561 0000090910000000 00000100
ACTIVE     00000   0129535  ::ffff:129.40.23...:5201 ::ffff:129.40.2...:40948 0000
SMCD 11000facb7f88561 0000090610000000 25000fadb7f88561 0000090710000000 00000100

```

You can use ISM if multiple subnets exist in your configuration on the same CEC. SMC-D is enhanced to remove the same subnet restriction by using SMC-Dv2.

SMC-R

SMC also can be enabled between different CPCs by using a RoCE card that allows remote direct memory access (RDMA) over the external network (SMC-R).

Example 4-39 shows how to check the RoCE device availability.

Example 4-39 Checking PCI devices

```

rdbkvm3:/home/lnxadmin/isos # lspci
0006:00:00.0 Ethernet controller: Mellanox Technologies MT27710 Family [ConnectX-4
Lx Virtual Function

```

The PNET ID in the OSA card is displayed. The PNET ID in the RoCE device also is shown. Both should display the same PNET ID.

4.6.8 Defining the MacVTap network

This section describes defining two MacVTap networks: one for OSA and another for HiperSockets.

The same configuration is used as in the SMC-D configuration. However, we use RoCE 2 instead ISM in this case.

Example 3-34 shows a similar example to Example 3-31 on page 69; however, the communication uses SMC-R in this case.

Example 4-40 Test results

```
[root@rdbkvm2 ~]# smcss -a
State      UID   Inode  Local Address          Peer Address          Intf Mode
ACTIVE     00000 0126538 ::ffff:129.40.23...:5201 ::ffff:129.40.2...:55590 0000 SMCR
ACTIVE     00000 0126537 ::ffff:129.40.23...:5201 ::ffff:129.40.2...:55588 0000 SMCR
LISTEN     00000 0126533 0.0.0.0:5201

[root@rdbkvm2 ~]# smcss -R
State      UID   Inode  Local Address          Peer Address          Intf Mode Role
IB-device  Port Linkid  GID                    Peer-GID
ACTIVE     00000 0126538 ::ffff:129.40.23...:5201 ::ffff:129.40.2...:55590 0000 SMCR SERV
mlx5_0    01    01      0000:0000:0000:0000:0000:ffff:8128:17df
fe80:0000:0000:0000:8017:9bff:fea8:a19b
ACTIVE     00000 0126537 ::ffff:129.40.23...:5201 ::ffff:129.40.2...:55588 0000 SMCR SERV
mlx5_0    01    01      0000:0000:0000:0000:0000:ffff:8128:17df
fe80:0000:0000:0000:8017:9bff:fea8:a19b
```

If multiple subnets exist between IBM z15 CPCS, SMC Version 2 (SMCv2) allows you to enable multiple IP subnet capability for SMC. This capability is enabled by updates to the underlying networking specifications for RoCE (RoCEv2) and the IBM Z ISM feature (ISMv2) along with updates to the related technologies.

4.6.9 Defining the MacVTap network

This section describes defining two MacVTap networks: one for OSA and another for HiperSockets.

MACVTap for OSA NICs

Instead of the use of the default network connectivity for the guests network address translation (NAT) connections, we chose MacVTap in bridge mode. This mode enables the guests a direct connection with the specified interface in the MacVTap network.

To configure the MacVTap network, we use the `virsh` command and an XML definition file. Example 4-41 shows our `macvtap-net.xml` network definition file.

Example 4-41 macvtap-net.xml

```
rdbkvm3:/home/lnxadmin # cat macvtap-net1.xml
<network>
  <name>macvtap-net1</name>
  <forward mode="bridge">
    <interface dev="bond0.8"/>
  </forward>
</network>
```

Example 4-42 shows the `virsh` command that is used to define a MacVTap network.

Example 4-42 virsh net-define command

```
rdbkvm3:/home/lnxadmin # virsh net-define macvtap-net1.xml
Network macvtap-net1 defined from macvtap-net1.xml
```

Example 4-43 shows how to set MacVTap-net persistence and start the network.

Example 4-43 virsh net-autostart and net.start command

```
rdbkvm3:/home/lnxadmin # virsh net-autostart macvtap-net1
Network macvtap-net1 marked as autostarted
rdbkvm3:/home/lnxadmin # virsh net-start macvtap-net1
Network macvtap-net1 started
```

MacVTap for HiperSockets NIC

The same steps that are used in 4.6.9, “Defining the MacVTap network” on page 122 are applied to the MacVTap HiperSockets definition. Example 4-44 shows the XML file that was created to define the HiperSockets NIC.

Example 4-44 macvtap-hsi.xml

```
rdbkvm3:/home/lnxadmin/isos # cat macvtap-hsi0.xml
<network>
  <name>macvtap-hsi0</name>
  <forward mode="bridge">
    <interface dev="hsi0"/>
  </forward>
</network>
```

4.6.10 Defining crypto adapters and domain

As described in 2.2.4, “Encryption considerations” on page 33, the Crypto Express card advantages can be used by the KVM hosts and VM guest.

It is important to check the compatibility list for Crypto Express adapters when SLES is used *before* beginning the installation. For more information about supported Crypto Express adapters with your version of SLES, see this [IBM Documentation web page](#).

To make the AP cards available to the KVM guests (see “Cryptography” on page 105), use the VFIO mediated device framework to assign cryptographic adapter resources to the device.

For this process, load the `vfio_ap` device driver by running the `modprobe vfio_ap` command. Then, add adapters 0x0 to the device, as shown in Example 4-45 and Example 4-46 on page 124.

Example 4-45 Enabling vfio_ap permanently

```
rdbkvm3:/etc # vim /etc/modules-load.d/vfio_ap.conf
rdbkvm3:/etc # cat /etc/modules-load.d/vfio_ap.conf
vfio_ap
```

Example 4-46 Preparing crypto usage

```
rdbkkvm3:/home/lxadmin/isos # modprobe vfio_ap
rdbkkvm3:/home/lxadmin/isos # echo 0x0 > /sys/bus/ap/apmask
rdbkkvm3:/home/lxadmin/isos # echo 0x0 > /sys/bus/ap/aqmask
```

Run the **lszcrypt** command to display information about the crypto adapters, as shown in Example 4-47.

Example 4-47 Displaying information about crypto adapters

```
rdbkkvm3:/home/lxadmin/isos # lszcrypt
CARD.DOMAIN TYPE MODE STATUS REQUESTS
-----
03          CEX7C CCA-Coproc online      1
06          CEX7C CCA-Coproc online
```

Assign AP queues to the KVM. Example 4-48 shows the procedure to assign the two crypto cards (03 and 06) and domain (0x1e) to the KVM host.

Example 4-48 Crypto for KVM host

```
rdbkkvm3:/home/lxadmin/isos # echo +0x03 > /sys/bus/ap/apmask
rdbkkvm3:/home/lxadmin/isos # echo +0x06 > /sys/bus/ap/apmask
rdbkkvm3:/home/lxadmin/isos # echo +0x1e > /sys/bus/ap/aqmask
```

Example 4-49 shows the verification of the crypto assignment to the KVM host.

Example 4-49 Verifying crypto assignment

```
rdbkkvm3:/home/lxadmin/isos # lszcrypt
CARD.DOMAIN TYPE MODE STATUS REQUESTS
-----
03          CEX7C CCA-Coproc online      2
03.001e    CEX7C CCA-Coproc online      2
06          CEX7C CCA-Coproc online      0
06.001e    CEX7C CCA-Coproc online      0
```

One way to make permanent configuration of the cryptos is running scripts by using the **insserv** command, which enables an installed system init script (“boot script”).

Example 4-50 Installing the insserv package

```
rdbkkvm3:/home/lxadmin # zypper install insserv-compat
```

We created the `sles_maintenance` service to run the scripts that are shown in Example 4-51 at start.

Example 4-51 Adding 01-crypto_enablement.sh script

```
rdbkkvm3:/etc/init.d/sles_maintenance.d/start.d # vim 01-crypto_enablement.sh
rdbkkvm3:/etc/init.d/sles_maintenance.d/start.d # chmod u+x
/etc/init.d/sles_maintenance.d/start.d/01-crypto_enablement.sh
rdbkkvm3:/etc/init.d/sles_maintenance.d/start.d # cat 01-crypto_enablement.sh
#!/bin/bash
# Freeing ap and aq queues for crypto enablement
echo Preparing crypto enviroment
echo 0x0 > /sys/bus/ap/apmask
echo 0x0 > /sys/bus/ap/aqmask
echo +0x03 > /sys/bus/ap/apmask
echo +0x06 > /sys/bus/ap/apmask
echo +0x1e > /sys/bus/ap/aqmask
```

For more information about configuring the `insserv`, see this [SUSE web page](#).

Example 4-52 shows how to generate a Universally Unique Identifier (UUID) for the mediated device, create the mediated device, and assign the crypto cards and crypto domains to it (for use and control).

Example 4-52 Generating a UUID for VM guest

```
rdbkkvm3:/home/lxadmin/isos # uuidgen
6d3b463c-4bf0-41f8-b22d-2750692431f6
rdbkkvm3:/home/lxadmin/isos#echo6d3b463c-4bf0-41f8-b22d-2750692431f6>/sys/device
s/vfio_ap/matrix/mdev_supported_types/vfio_ap-passthrough/create

rdbkkvm3:/home/lxadmin # echo 6d3b463c-4bf0-41f8-b22d-2750692431f6 >
/sys/devices/vfio_ap/matrix/mdev_supported_types/vfio_ap-passthrough/create
rdbkkvm3:/home/lxadmin # echo 0x03 >
/sys/devices/vfio_ap/matrix/6d3b463c-4bf0-41f8-b22d-2750692431f6/assign_adapter
rdbkkvm3:/home/lxadmin # echo 0x06 >
/sys/devices/vfio_ap/matrix/6d3b463c-4bf0-41f8-b22d-2750692431f6/assign_adapter
rdbkkvm3:/home/lxadmin # echo 0x001f >
/sys/devices/vfio_ap/matrix/6d3b463c-4bf0-41f8-b22d-2750692431f6/assign_domain
rdbkkvm3:/home/lxadmin # echo 0x001f >
/sys/devices/vfio_ap/matrix/6d3b463c-4bf0-41f8-b22d-2750692431f6/assign_control_domain
```

The procedure that is shown in Example 4-52 must be done for each domain that is used by a VM. In our lab environment, we used domains 31 and 32.

Example 4-53 shows how to verify the mediated device crypto assignment.

Example 4-53 Verifying mediated device crypto assignment

```
rdbkkvm3:/home/lxadmin # cat
/sys/devices/vfio_ap/matrix/6d3b463c-4bf0-41f8-b22d-2750692431f6/matrix
03.001f
06.001f
```

To make the `vfiopass` persistent, you must install the `mdevctl` package and then, run the commands that are shown in Example 4-54 by using the UUID, which is the adapter and domains that are used for the mediated device.

Notes: Consider the following points:

- ▶ Mediated devices can be configured manually by using `sysfs` operations.
- ▶ The `mdevctl` utility is used for managing and persisting devices in the mediated device framework of the Linux kernel. For more information, see this [GitHub web page](#).
- ▶ The `driverctl` device driver is a control utility for Linux. For more information, see this [GitLab web page](#).

Example 4-54 Making vfiopass mediated device persistent

```
rdbrkvm3:/home/lxadmin # mdevctl define --uuid 6d3b463c-4bf0-41f8-b22d-2750692431f6
--parent matrix --type vfiopass-passthrough
rdbrkvm3:/home/lxadmin # mdevctl modify --uuid 6d3b463c-4bf0-41f8-b22d-2750692431f6
--addattr=assign_adapter --value=0x03
rdbrkvm3:/home/lxadmin # mdevctl modify --uuid 6d3b463c-4bf0-41f8-b22d-2750692431f6
--addattr=assign_adapter --value=0x06
rdbrkvm3:/home/lxadmin # mdevctl modify --uuid 6d3b463c-4bf0-41f8-b22d-2750692431f6
--addattr=assign_domain --value=0x001f
rdbrkvm3:/home/lxadmin # mdevctl modify --uuid 6d3b463c-4bf0-41f8-b22d-2750692431f6
--addattr=assign_control_domain --value=0x001f
rdbrkvm3:/home/lxadmin # mdevctl start --uuid 6d3b463c-4bf0-41f8-b22d-2750692431f6
rdbrkvm3:/home/lxadmin # mdevctl modify --uuid 6d3b463c-4bf0-41f8-b22d-2750692431f6
--auto rdbrkvm3:/home/lxadmin # mdevctl list
6d3b463c-4bf0-41f8-b22d-2750692431f6 matrix vfiopass-passthrough (defined)
```

Notes: Consider the following points:

- ▶ The mediated device must be started *after* the host is restarted.
- ▶ The `--auto` parameter that is shown in Example 4-54 triggers only autostart after a system restart if the user sets the `apmask` and `aqmask` correctly *and* loads the `vfiopass` driver.

4.7 Deploying VMs on KVM

In this section, we describe the deploying VMs in the KVM environment. Creating a VM can be done by using several methods. This section describes the `virt-install` command and `virsh` tools.

4.7.1 Creating QCOW2 disk image file

As described in “Disk” on page 104, QCOW2 files are used to create the VM disks.

Example 4-55 shows the command that is used to create a 10 GB QCOW2 file.

Example 4-55 Creating qcow2 image file

```
rdbkvm3:/home/lnxadmin/isos # qemu-img create -f qcow2 kvm3guest01_vol001.img 10G
Formatting 'kvm3guest01_vol001.img', fmt=qcow2 cluster_size=65536 extended_l2=off
compression_type=zlib size=10737418240 lazy_refcounts=off refcount_bits=16
```

4.7.2 Installing a new guest by using virt-install

The `virt-install` is a command-line tool that is used to create VMs on KVM that use the `libvirt` hypervisor management library.

Example 4-56 shows how to install a VM by using the `virt-install` command.

Example 4-56 Creating VM guest by using virt-install command

```
virt-install --name kvm3guest01 --vcpus 2 --memory 4000 --os-variant sles13sp5
--disk path=/home/lnxadmin/isos/kvm3guest01_vol001.img --network
network:macvtap-net1 --cdrom /home/lnxadmin/SLE-15-SP3-Full-s390x-GM-Media1.iso
--graphics none
```

Consider the following points:

- ▶ The `--name` parameter specifies the name of the VM guest.
- ▶ The `--nographics` parameter must be specified to disable the graphics installation.
- ▶ The `--memory` parameter specifies the amount of memory (RAM) that is allocated to the VM (expressed in megabytes).
- ▶ The `--vcpus` parameter specifies how many vcpus are assigned to the VM.
- ▶ The `--os-variant` parameter specifies which type of operating system is to be installed; this option is highly recommended when importing a disk image. If it is not provided, the performance of the VM that is created is negatively affected. Run the `osinfo-query os` command for a full list of available operating systems.
- ▶ The `--disk` parameter specifies the media to use as storage for the VM guest (kvm3guest01 uses QCOW2 files). If the file was preallocated, specify the `--import` parameter. Otherwise, you can omit the `--import` parameter and include the new file path by using the parameters format and size to allocate the file during the installation.
- ▶ The `--network` parameter specifies the network options for the VM guest. In this case, we are connecting the guest to MacVTap-net that was created as described in “Defining the MacVTap network” on page 122.

- For the installation source, we used an .iso file that uses the --cdrom parameter. You also can install from other sources, such as an FTP server

After the command is issued (see Example 4-56 on page 127), the VM installation begins, as shown in Figure 4-6.

```
Starting install...
Running text console command: virsh --connect qemu:///system console
kvm3guest01
Connected to domain 'kvm3guest01'
Escape character is ^] (Ctrl + ])
[ 0.031118] Linux version 5.3.18-57-default (geeko@buildhost) (gcc version
7.5.0 (SUSE Linux)) #1 SMP Wed Apr 28 10:54:41 UTC 2021 (ba3c2e9)
[ 0.031120] setup: Linux is running under KVM in 64-bit mode
[ 0.032419] setup: The maximum memory size is 4000MB
[ 0.032444] numa: NUMA mode: plain
[ 0.032491] cpu: 2 configured CPUs, 0 standby CPUs
[ 0.032567] Write protected kernel read-only data: 10620k
[ 0.032608] Zone ranges:
[ 0.032609]   DMA      [mem 0x0000000000000000-0x000000007fffffff]
[ 0.032610]   Normal  [mem 0x0000000080000000-0x00000000f9fffffff]
[ 0.032612] Movable zone start for each node
[ 0.032612] Early memory node ranges
[ 0.032613]   node 0: [mem 0x0000000000000000-0x00000000f9fffffff]
[ 0.032615] Initmem setup node 0 [mem 0x0000000000000000-0x00000000f9fffffff]
[ 0.068688] percpu: Embedded 33 pages/cpu s98048 r8192 d28928 u135168
[...]
>>> SUSE Linux Enterprise 15 SP3 installation program v7.0.30.3 (c) 1996-2020
SUSE LLC on IBM z15 <<<

Starting udev... [ 9.512535] SCSI subsystem initialized
[ 9.514322] alua: device handler registered
[ 9.532998] emc: device handler registered
[ 9.551702] rdac: device handler registered
[ 10.005060] VFIO - User Level meta-driver version: 0.3
[ 10.009286] vfio_ccw: externally supported module, setting X kernel taint
flag.
[ 10.056176] virtio_blk virtio2: [vda] 20971520 512-byte logical blocks (10.7
GB/10.0 GiB)
```

Figure 4-6 VM guest installation process by using virt-install command

After the restart at the end of the installation process, we delete the parameters that are specified in the --boot section. Example 4-57 shows how to edit the VM guest domain to delete <kernel></kernel>, <initrd></initrd>, and <cmdline></cmdline>.

Example 4-57 Editing VM guest domain

```
rdbkvm3:/var/lib/libvirt/images # virsh edit kvm3guest01
<domain type='kvm'>
  <name>kvm3guest01</name>
[...]
<os>
  type arch='s390x' machine='s390-ccw-virtio-3.1'>hvm</type>
  <kernel>/var/lib/libvirt/images/s15p1-kernel.boot</kernel>
```



```

    <initrd>/var/lib/libvirt/images/s15p1-initrd.boot</initrd>
    <cmdline>HostIP=9.76.61.32/24 Hostname=kvmguest03 Gateway=9.76.61.1 Layer2=1
Install=ftp://itso:itso1cpo@9.76.56.32/SLES15SP1IDVD1/ Manual=1</cmdline>
    <boot dev='hd' />
  </os>
  [...]
</domain>
Domain kvmguest03 XML configuration edited.

```

Finally, recycle the VM guest (by using the **virsh destroy** and **virsh start** commands) to complete the changes.

For more information about the **virt-install** command, see this [SUSE Documentation web page](#).

4.7.3 Cloning a guest by using Virsh

Virsh is a command-line tool that is used to manage VM guests and the hypervisor. It also uses the libvirt hypervisor management library. In this section, we show how to clone a VM from a previous image installation base.

Example 4-58 on page 129 shows the first task. Copy the QCOW2 file `kvm3guest01_vo1001.img` to `kvm3guest02_vo1001.img`.

Example 4-58 Copying the QCOW2 file

```

rdbkkvms:/var/lib/libvirt/images # cp kvm3guest01_vo1001.img
kvm3guest02_vo1001.img

```

Run the **dumpxml** command to return the guest VM's configuration file. As shown in Example 4-59, we obtain the XML configuration file `kvmsvm02.xml` from the VM guest, `kvmsvm01`.

Example 4-59 Creating the guest configuration file

```

rdbkkvm3: /var/lib/libvirt/images # virsh dumpxml kvm3guest01 > kvm3guest02.xml

```

Because the VM guest is to be cloned, complete the following steps to edit `kvm3guest02.xml`:

1. Change the VM name in the file from:

```
<name>kvm3guest01</name>
```

to:

```
<name>kvm3guest02</name>.
```

2. Delete the following UUID assignment statement:

```
<uuid>b4b9e0fd-b8e5-4b95-9192-9e385f1e4864</uuid>
```

3. Change the source file of QCOW2 disk from:

```
<source file='/var/lib/libvirt/images/kvm3guest01_vo1001.img' />
```

to:

```
<source file='/var/lib/libvirt/images/kvm3guest02_vo1001.img' />
```

4. Complete the following steps in the `<interface type='direct'>` section:

- a. Delete the MAC address statement: `<mac address='52:54:00:6b:8d:f7' />`.

- b. Delete the target device statement: `<target dev='macvtap1'/>`.

All deleted information is dynamically generated when we use the **virsh define** command.

The `kvm3guest02` guest is defined as shown in Example 4-60.

Example 4-60 Defining the `kvm3guest2` guest

```
rdbkkvm3:/home/lnxadmin/isos # virsh define kvm3guest02.xml
Domain kvm3guest02 defined from kvm3guest02.xml
```

To start the new cloned guest, run **virsh start `kvm3guest02`**.

You must change the basic parameters on the new guest, such as the IP address and hostname.

Another way to clone guests is by using the **virt-clone** command, as shown in Example 4-61. The guest *must* be shut down to be cloned.

Example 4-61 Cloning the `kvm3guest01` guest

```
rdbkkvm3:/home/lnxadmin/isos # virt-clone --original kvm3guest01 --name kvm3guest02 --file
/home/lnxadmin/isos/kvm3guest02_vol001.img
Allocating 'kvm3guest02_vol001.img'
| 10 GB 00:00:01

Clone 'kvm3guest02' created successfully.
```

Consider the following points:

- ▶ The `--original` statement indicates the name of the guest (domain) to be cloned.
- ▶ The `--name` statement indicates the name of the new guest (domain) to be created.
- ▶ The `--file` statement indicates the location of the new `qcow2` that is to be allocated for the new guest.

4.7.4 Adding HiperSockets to the VM guest

To add an NIC, a VM is needed to shut down the guest and edit the domain definition. In this example, we use a vNIC, `macvtap-hsi`, which targets the `hsi0` HiperSockets interface.

Example 4-62 shows the command that is used to edit the VM domain definition in XML format.

Example 4-62 Edit domain definition

```
rdbkkvm3:/home/lnxadmin # virsh edit kvm3guest01
Domain kvm3guest01 XML configuration edited.
```

You also must add the definition that is shown in Example in the `<devices>` `</devices>` section.

Example 4-63 Interface definition

```
<devices>
  <interface type='network'>
    <source network='macvtap-hsi0'/>
    <model type='virtio'/>
```

```
</interface>
</devices>
```

After the domain starts, the VM shows the new interface and that the domain definition was updated, as shown in Example 4-64.

Example 4-64 interface verification

At the VM level:

```
kvm3guest01:/home/lrxadmin # ip a show eth1
2: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen
1000
    link/ether 52:54:00:ae:00:3f brd ff:ff:ff:ff:ff:ff
    altname enc6
```

At the KVM host:

```
rdbkvm3:/home/lrxadmin # virsh dumpxml kvm3gues01
[...]
<interface type='direct'>
  <mac address='52:54:00:ae:00:3f'/>
  <source network='macvtap-hsi0' portid='75e0d07c-a144-4d73-9257-a7c24e251f7c'
dev='hsi0' mode='bridge'/>
  <target dev='macvtap4'/>
  <model type='virtio'/>
  <alias name='net1'/>
  <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0006'/>
</interface>
[...]
```

4.7.5 Adding space to guest from ECKD DASD

To add space to a VM in our lab, we added a full volume DASD as a virtio device. You can also add space by using a logical volume from an LVM pool.

The first step is formatting the volume on the host as the example shows for the ECKD device 0.0.914B. For this task, we check whether the device is available for the LPAR and enabled it before the formatting process (see Example 4-65 and Example 4-66 on page 132).

Example 4-65 DASD formatting

```
rdbkvm3:/home/lrxadmin/isos # chzdev -e dasd 0.0.904b
ECKD DASD 0.0.914b configured
rdbkvm3:/home/lrxadmin/isos # dasdfmt -b 4096 /dev/disk/by-path/ccw-0.0.914b -p
--label=0x914B
Drive Geometry: 60102 Cylinders * 15 Heads = 901530 Tracks
Device Type: Thinly Provisioned
```

I am going to format the device /dev/disk/by-path/ccw-0.0.914b in the following way:

```
Device number of device : 0x914b
Labelling device         : yes
Disk label               : VOL1
Disk identifier          : 0X914B
Extent start (trk no)   : 0
```

```
Extent end (trk no)    : 1
Compatible Disk Layout : yes
Blocksize              : 4096
Mode                  : Quick
Full Space Release     : yes
```

WARNING:

Disk /dev/disk/by-path/ccw-0.0.914b is online on operating system instances in 15 different LPARs.

Ensure that the disk is not being used by a system outside your LPAR.

Note: Your installation might include z/VM systems that are configured to automatically vary on disks, regardless of whether they are subsequently used.

---> ATTENTION! <---

All data of that device will be lost.

Type "yes" to continue, no will leave the disk untouched: yes

Releasing space for the entire device...

Skipping format check due to thin-provisioned device.

Formatting the first two tracks of the device.

Finished formatting the device.

Rereading the partition table... ok

For get more i/o performance on the virtual block devices we can configure one or more I/O threads for the virtual server and each virtual block device can use one of these I/O threads.

Example 4-66 Creating I/O thread for guest

```
rdbkkvm3:/home/lrxadmin/isos # virsh iothreadadd --domain kvm3guest01 --id 1
--live
rdbkkvm3:/home/lrxadmin/isos # virsh iothreadadd --domain kvm3guest01 --id 1
--config
```

Now, we can define and attach to the guest the new formatted DASD. Remember to always format the DASD on the host, but create the partitions at the guest level if you plan to assign the hole disk to the guest (see Example 4-67).

Example 4-67 Defining the virtual block device .xml and attach to the guest

```
rdbkkvm3:/home/lrxadmin/isos # vim kvm3guest01_dasd01.xml
rdbkkvm3:/home/lrxadmin/isos # cat kvm3guest01_dasd01.xml
<disk type="block" device="disk">
<driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
<source dev="/dev/disk/by-path/ccw-0.0.914b"/>
<target dev="vdb" bus="virtio"/>
<address type="ccw" cssid="0xfe" ssid="0x0" devno="0x00a8"/>
</disk>
rdbkkvm3:/home/lrxadmin/isos # virsh attach-device kvm1guest01
kvm3guest01_dasd01.xml --persistent
Device attached successfully
```

Verification commands

In KVM, we can verify the use of the virtio device (see Example 4-68 and Example 4-69).

Example 4-68 Verifying the use of virtio device

```
rdbkkvm3:/home/lnxadmin # virsh domblklist kvm3guest01
Target  Source
-----
vda     /home/lnxadmin/isos/kvm3guest01_vol001.img
vdb     /dev/disk/by-path/ccw-0.0.914b
sda     -
```

Example 4-69 Verifying the device availability on the guest

```
kvm3guest01:/home/lnxadmin # lsblk
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
sr0   11:0    1 1024M 0 rom
vda   253:0    0  10G  0 disk
??vda1 253:1    0  300M  0 part /boot/zip1
??vda2 253:2    0   9.7G  0 part /
vdb   253:16   0  41.3G  0 disk
```

4.7.6 Adding DASD space to guest as a VFIO device

Another way to add DASD to the guest is by using a VFIO pass-through device, which allows the guest to control the whole DASD as a direct device. To add a DASD to the guest, you must bring its subchannel under control of the `vfio_ccw` device driver, create a mediated device for the DASD and then, assign the mediated device to the guest (see Example 4-70 - Example 4-75 on page 134).

Example 4-70 Checking the device sub-channel

```
rdbkkvm3:/home/lnxadmin # lscss -a | grep 904b
Device  Subchan.  DevType  CU  Type  Use  PIM  PAM  POM  CHPIDs
-----
0.0.904b 0.0.24e9  3390/0c  3990/e9      f0  f0  ff  40424143 00000000
```

Example 4-71 Unbinding the device from host and creating the mediated device for DASD

```
rdbkkvm3:/home/lnxadmin # echo 0.0.904b > /sys/bus/ccw/drivers/dasd-eckd/unbind
rdbkkvm3:/home/lnxadmin # echo 0.0.26e9 >
/sys/bus/css/devices/0.0.26e9/driver/unbind
rdbkkvm3:/home/lnxadmin # echo 0.0.26e9 > /sys/bus/css/drivers/vfio_ccw/bind
rdbkkvm3:/home/lnxadmin # uuidgen
fef02213-6b37-4434-9158-c4105c8d2b6f
[root@rdbkkvm1 network-scripts]# echo fef02213-6b37-4434-9158-c4105c8d2b6f >
/sys/bus/css/devices/0.0.26e9/mdev_supported_types/vfio_ccw-io/create
```

Example 4-72 Adding the mediated device to the definition in the device section

```
rdbbkvm3:/home/lnxadmin # virsh edit kvm3guest02
  <hostdev mode="subsystem" type="mdev" model="vfio-ccw">
    <source>
      <address uuid="fef02213-6b37-4434-9158-c4105c8d2b6f"/>
    </source>
    <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x00b1"/>
  </hostdev>
```

Example 4-73 Checking at the guest level

```
kvm3guest02:/home/lnxadmin # lscss
Device  Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
-----
0.0.0002 0.0.0000  0000/00 3832/08 yes  80  80  ff  00000000 00000000
0.0.0003 0.0.0001  0000/00 3832/03 yes  80  80  ff  00000000 00000000
0.0.0000 0.0.0002  0000/00 3832/02 yes  80  80  ff  00000000 00000000
0.0.0001 0.0.0003  0000/00 3832/01 yes  80  80  ff  00000000 00000000
0.0.00b1 0.0.0004  3390/0c 3990/e9 yes  f0  f0  ff  40424143 00000000
0.0.0004 0.0.0005  0000/00 3832/05 yes  80  80  ff  00000000 00000000
0.0.0005 0.0.0006  0000/00 3832/04 yes  80  80  ff  00000000 00000000
```

Example 4-74 Enabling the DASD on the guest

```
kvm3guest02:/home/lnxadmin # chzdev -e dasd 0.0.00b1
ECKD DASD 0.0.00b1 configured
Note: The initial RAM-disk must be updated for these changes to take effect:
      - ECKD DASD 0.0.00b1
```

Example 4-75 Verifying the device availability at guest level

```
kvm3guest02:/home/lnxadmin # lsdasd
Bus-ID  Status  Name      Device  Type          BlkSz  Size      Blocks
=====
0.0.00b1 active  dasda     94:0    ECKD (ESE)   4096   42259MB  10818360
kvm3guest02:/home/lnxadmin # lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0         11:0    1 1024M 0 rom
dasda       94:0    0  41.3G 0 disk
??dasda1    94:1    0  41.3G 0 part
vda         253:0   0   10G  0 disk
??vda1      253:1   0   300M 0 part /boot/zip1
??vda2      253:2   0    9.7G 0 part /
```

To make the `vfio_ccw` persistent, the `mdevctl` and `driverctl` packages must be installed and the commands that are shown in Example 4-76 on page 135 run by using the UUID and the subchannel that are selected for this device.

Notes: Consider the following points:

- ▶ Mediated devices can be configured manually by using `sysfs` operations.
- ▶ The `mdevctl` utility is used for managing and persisting devices in the mediated device framework of the Linux kernel. For more information, see this [GitHub web page](#).
- ▶ The `driverctl` device driver is a control utility for Linux. For more information, see this [GitLab web page](#).

Example 4-76 Making vfio_ccw mediated device persistent

```
rdbkkvm3:/home/lnxadmin # driverctl -b css set-override 0.0.24e9 vfio_ccw
rdbkkvm3:/home/lnxadmin # mdevctl define -u fef02213-6b37-4434-9158-c4105c8d2b6f
-p 0.0.24e9 -t vfio_ccw-io
rdbkkvm3:/home/lnxadmin # mdevctl start --uuid
fef02213-6b37-4434-9158-c4105c8d2b6f
rdbkkvm3:/home/lnxadmin # mdevctl modify --uuid
fef02213-6b37-4434-9158-c4105c8d2b6f --auto
rdbkkvm3:/home/lnxadmin # mdevctl list
fef02213-6b37-4434-9158-c4105c8d2b6f 0.0.24e9 vfio_ccw-io (defined)
```

4.7.7 Adding LUNs when FCP SCSI storage is used

To add space to a VM, you can map a target LUN. In this case, you can choose an available LUN to identify the device-ID that we present to the VM.

As described in 2.2.2, “Storage considerations” on page 27, the following options are available:

- ▶ Entire disk (LUN or ECKD DASD)
- ▶ Partition of the disk
- ▶ Logical volume

Map the device ID by using the multipath ID. Avoid the use of multipath-friendly names.

Example 4-77 shows how to identify the target LUN.

Example 4-77 Identifying the LUN example

```
rdbkkvm3:/home/lnxadmin/isos # multipath -ll | grep
36005076309ffd145000000000000000c
36005076309ffd145000000000000000c dm-11 IBM,2107900
```

Example 4-78 shows the identification by device ID.

Example 4-78 Device mapper mpath identification by device ID

```
rdbkkvm3:/dev/disk/by-id # ls | grep 36005076309ffd145000000000000000c
dm-name-36005076309ffd145000000000000000c
dm-uuid-mpath-36005076309ffd145000000000000000c
scsi-36005076309ffd145000000000000000c
```

After identifying the target LUN and the device ID for our lab environment, the target disk is:

```
/dev/disk/by-id/dm-uuid-mpath-36005076309ffd145000000000000000c
```

With this information available, the next step is to create an XML file to attach the disk, as shown in Example 4-79.

Example 4-79 Device mapper mpath identification by device ID

```
rdbkkvm3:/home/lnxadmin/isos # vim kvm3guest01_block1.xml
rdbkkvm3:/home/lnxadmin/isos # cat kvm3guest01_block1.xml
<disk type="block" device="disk">
  <driver name="qemu" type="raw" cache="none" io="native"/>
  <source
dev="/dev/disk/by-id/dm-uuid-mpath-36005076309ffd145000000000000000c"/>
```

```
<target dev="vdb" bus="virtio"/>
</disk>
```

Attach disk to the VM guest, as shown in Example 4-80.

Example 4-80 Attaching disk to kvmsvm01 guest

```
rdbkkvm3:/home/lnxadmin/isos # virsh attach-device kvm3guest01
kvm3guest01_block1.xml --persistent
Device attached successfully
```

In guest kvmsvm01:

```
rdbkkvm3:/home/lnxadmin/isos # virsh attach-device kvm3guest01
kvm3guest01_block1.xml --persistent
Device attached successfully
```

Validate that the host and guest are attached to the disk, as shown in Example 4-81.

Example 4-81 Verifying that the host and guest are attached to the disk

From KVM host:

```
rdbkkvm3:/home/lnxadmin/isos # virsh domblklist kvm3guest01
Target    Source
-----
vda       /var/lib/libvirt/images/kvm3guest01_vol001.img
vdb       /dev/disk/by-id/dm-uuid-mpath-36005076309ffd145000000000000000c
```

From kvm3guest01 guest:

```
kvm3guest01:/home/lnxadmin # lsblk
NAME     MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda      254:0   0   10G  0 disk
??vda1  254:1   0   300M  0 part /boot/zip1
??vda2  254:2   0    9.7G  0 part /
vdb      254:16  0    40G  0 disk
```

4.7.8 Adding cryptography support to the VM guest

As described in 4.6.10, “Defining crypto adapters and domain” on page 123, the crypto adapters and domain were defined. The AP queues were then assigned for use by KVM. The `vfiop` mediated device was created to enable the assignment of the crypto device to a VM guest.

Complete the following steps to add cryptography support to the VM guest:

1. In the VM domain definition, edit the XML file (see Example 4-82). Locate the `<devices>` section and add the `<hostdev>` section, as shown in Example 4-82.

Example 4-82 Editing VM definitions by using virsh

```
rdbkkvm3:/var/lib/libvirt/images # virsh edit kvm3guest01
Domain kvm3guest01 XML configuration edited.
```

Locate the <devices> section and add the <hostdev> section, as shown in Example 4-83.

Example 4-83 Mediated device definition

```
<hostdev mode='subsystem' type='mdev' managed='no' model='vfio-ap'>
  <source>
    <address uuid='6d3b463c-4bf0-41f8-b22d-2750692431f6' />
  </source>
</hostdev>
```

The true random number generator (TRNG) feature can be used to generate random numbers (see Example 4-84). For more information, see Chapter 2, “Planning for the Kernel-based Virtual Machine host and guest” on page 21.

Example 4-84 Statement to use TRNG

```
<rng model='virtio'>
  <backend model='random'>/dev/trng</backend>
</rng>
```

2. Recycle the VM and verify the definitions by running the commands that are shown in Example 4-85.

Example 4-85 Verification commands

```
rdbkkvm3:/home/lnxadmin/isos # cat /sys/devices/virtual/misc/trng/byte_counter
trng: 32
hwrng: 32
arch: 472720
total: 472784
```

On the guest, we verify the crypto availability:

```
kvm3guest01:/home/lnxadmin # lszcrypt
CARD.DOMAIN TYPE MODE STATUS REQUESTS
-----
03          CEX7C CCA-Coproc online      1
03.001f     CEX7C CCA-Coproc online      1
06          CEX7C CCA-Coproc online      0
06.001f     CEX7C CCA-Coproc online      0
```

Upon completion of these steps, the crypto card is available to be used in the entire environment, including the KVM host and the VMs.

For more information, see *Configuring Crypto Express Adapters for KVM Guests*, [SC34-7717](#).

4.7.9 Using the Integrated Accelerator for zEnterprise Data Compression

The Integrated Accelerator for zEnterprise Data Compression (zEDC) with the IBM® z15 replaces the zEDC Express adapter with on-chip compression, which provides increased throughput and capacity. It also reduces the cost of storing, processing, and transporting data.

The acceleration with the on-chip Integrated Accelerator for zEDC is available to applications that use zlib or gzip in user space and to the kernel zlib.

To check whether your platform can use the Integrated zEDC, you must check whether the `df1t` feature is available by using the command that is shown in Example 4-86.

Example 4-86 Checking the `df1t` feature

```
root@kvm3guest01:/home/rdbkuser1# lscpu | grep df1t
Flags:                esan3 zarch stfle msa ldisp eimm dfp edat etf3eh
highgrps te vx vxd vxe gs vxe2 vxp sort df1t
```

In our lab, the `df1t` feature is available because we use an IBM z15. To use this feature, you must update the `DFLTCC_LEVEL_MASK` (see Table 4-5). For more information, see the [IBM Documentation web page](#).

Table 4-5 Compression statistics

Compression level	Compression ratio	Elapsed time	Total CPU Time
0x0000	91.6%	1m24.825s	1m18.283s
0x0002	91.6%	1m18.038s	1m17.794s
0x007E	88.9%	0m14.866s	0m1.737s
0x01FF	88.9%	0m16.419s	0m1.757.s

The zEDC compression exercises and statistics are listed in Table 4-6.

Table 4-6 zEDC compression exercises and statistics

Exercises:
<pre>kvm3guest01:/home/lnxadmin # export DFLTCC_LEVEL_MASK=0x0000 kvm3guest01:/home/lnxadmin # time gzip -v -c operlog.txt > operlog1_nohw.gz operlog.txt: 91.6% real 1m24.825s user 1m16.746s sys 0m1.537s kvm3guest01:/home/lnxadmin # export DFLTCC_LEVEL_MASK=0x0002 kvm3guest01:/home/lnxadmin # time gzip -v -c operlog.txt > operlog1_lvl_0x0002.gz operlog.txt: 91.6% real 1m18.038s user 1m16.298s sys 0m1.496s root@kvm1guest1:/home/rdbkuser1# export DFLTCC_LEVEL_MASK=0x007e kvm3guest01:/home/lnxadmin # time gzip -v -c operlog.txt > operlog1_lvl_0x007e.gz operlog.txt: 88.9% real 0m14.866s user 0m0.430s sys 0m1.307s root@kvm1guest1:/home/rdbkuser1# export DFLTCC_LEVEL_MASK=0x01ff kvm3guest01:/home/lnxadmin # time gzip -v -c operlog.txt > operlog1_lvl_0x01ff.gz operlog.txt: 88.9% real 0m16.419s user 0m0.430s sys 0m1.327s</pre>

For more information about Integrated Accelerator for zEDC, see this [IBM Documentation web page](#).



Preparing the Ubuntu Kernel-based Virtual Machine environment for virtual machine use

This chapter describes the installation process for Ubuntu on LPAR, prepare it as a Kernel-based Virtual Machine (KVM) host, and deploy the KVM guests.

This chapter includes the following topics:

- ▶ 5.1, “Defining the target configuration” on page 142
- ▶ 5.2, “Preparing the infrastructure” on page 144
- ▶ 5.3, “Collecting information” on page 146
- ▶ 5.4, “Installing Ubuntu on an LPAR as a KVM host” on page 151
- ▶ 5.5, “Preparing the host for virtualization” on page 152
- ▶ 5.6, “Configuring the KVM host” on page 154
- ▶ 5.7, “Deploying virtual machines on KVM” on page 173

5.1 Defining the target configuration

To prepare the environment for the workloads that run in the virtual machines (VMs), it is recommended to build an installation plan. For more information, see 2.2, “Planning resources for KVM guests” on page 26.

This section provides the instructions to configure and deploy a basic KVM environment on Ubuntu 21.10 LTS.

5.1.1 Logical View

The Logical View of our lab environment that is used in this book is shown in Figure 5-1. This view provides an overview of the entire environment and can be built during the planning phase. For more information, see Chapter 2, “Planning for the Kernel-based Virtual Machine host and guest” on page 21.

The following networks types for guests are available:

- ▶ External network through the MacVTap network
- ▶ Internal CPC network through the HiperSocket MacVTap network

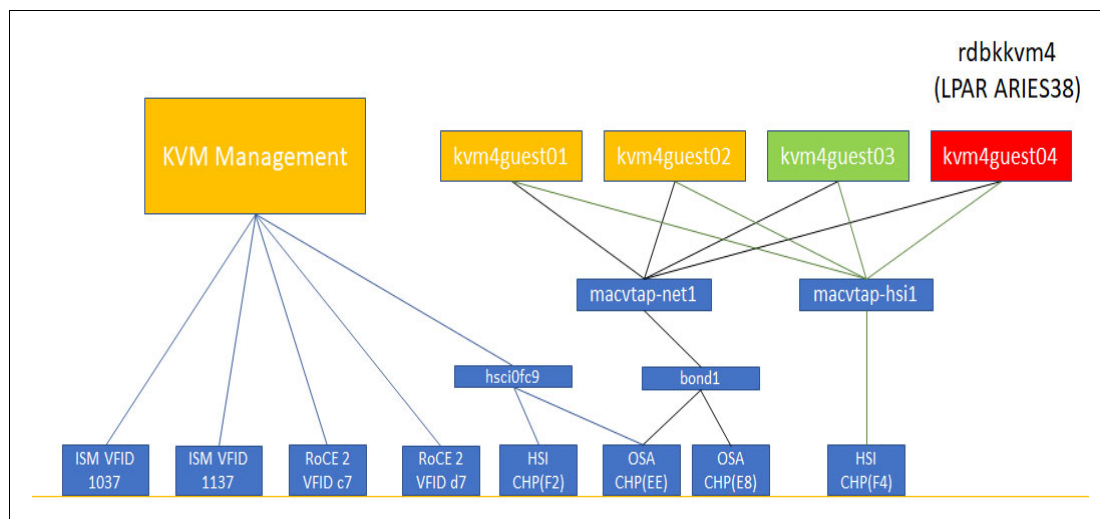


Figure 5-1 Ubuntu Logical view

The KVM host can access the following networks:

- ▶ HiperSockets network through the HSI (CHPID F4) interface.
- ▶ Internal Shared Memory or ISM (SMC-D VFID 1037 and 1137), as described in Chapter 2, “Planning for the Kernel-based Virtual Machine host and guest” on page 21.
- ▶ RoCE network (SMC-R, VFID C8 and D8), as described in Chapter 2, “Planning for the Kernel-based Virtual Machine host and guest” on page 21.
- ▶ External network through the OSA network interface card (NIC).

5.1.2 Physical resources

Figure 5-2 shows the hardware and connectivity setup, which includes the following components:

- ▶ One IBM z15 platform with four logical partitions (LPARs)
- ▶ Two OSA cards that are connected to a LAN
- ▶ Four FICON Express16SA+ cards for connection to the ECKD DASD on IBM DS8900F storage box
- ▶ One FTP server
- ▶ Two HiperSockets defined CHIPDs
- ▶ Two ISM defined as SMC-D
- ▶ Two RoCE cards as SMC-R
- ▶ Two CryptoExpress cards

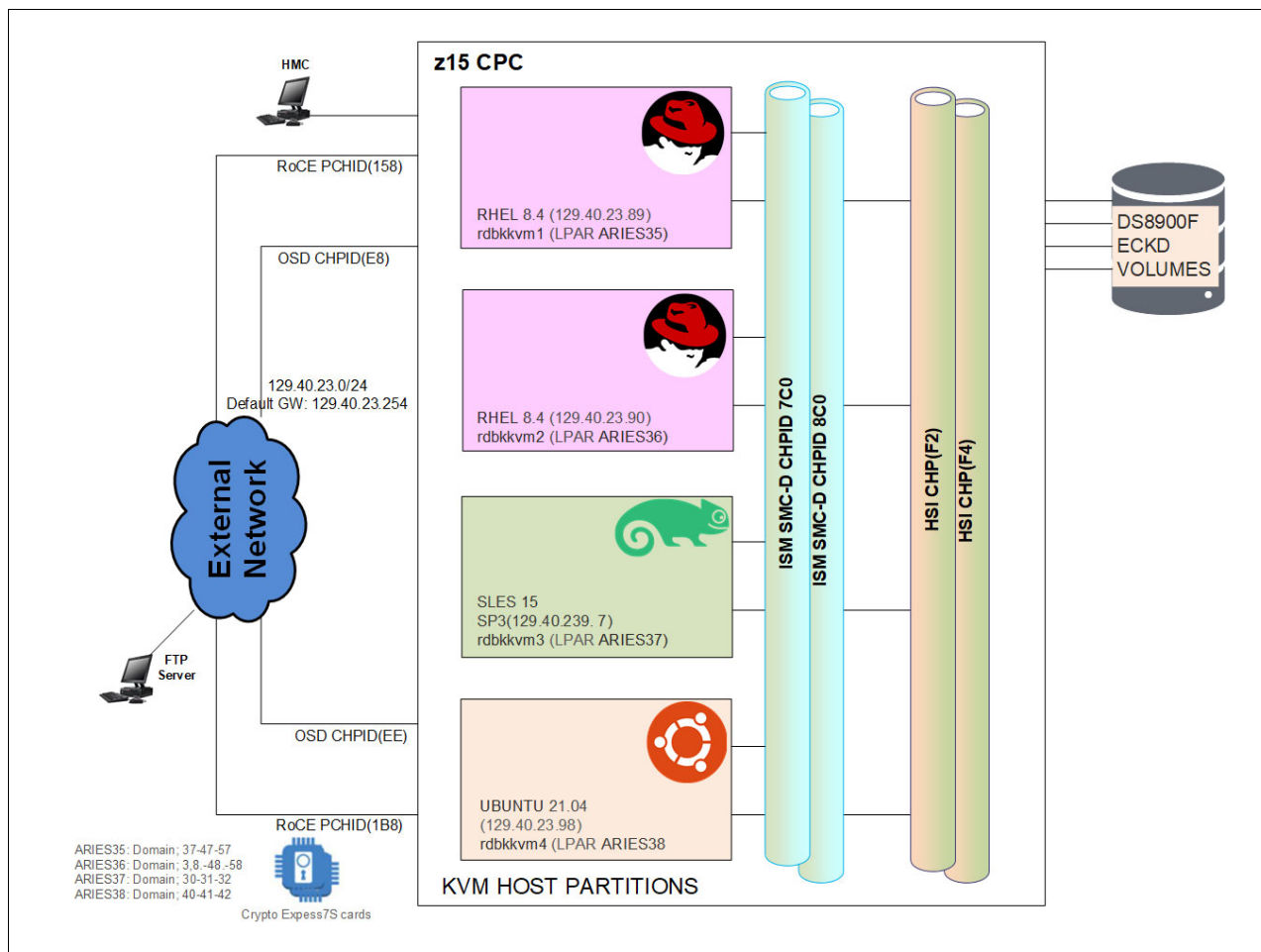


Figure 5-2 Ubuntu physical resources

All LPARs can access all resources. Our lab environment includes the following LPARS:

- ▶ ARIES35: For RHEL
- ▶ ARIES36: For RHEL
- ▶ ARIES37: For SLES
- ▶ ARIES38: For Ubuntu

Note: This chapter is focused on the ARIES38 LPAR for the Ubuntu implementation.

5.1.3 Software resources

The Ubuntu version that was used in our lab environment is 21.10 LTS, which is latest supported version for IBM Z. It is important to know that the operating system architecture of the Z platform is s390x and the Linux packages must be based on this architecture.

For more information about Ubuntu-supported versions on IBM Z, see this [IBM Documentation web page](#).

For KVM virtualization beyond the operating system, the virtualization package is required for the KVM host. For more information, see this [Ubuntu Documentation web page](#).

5.2 Preparing the infrastructure

The IT infrastructure planning depends on many factors, as described in Chapter 2, “Planning for the Kernel-based Virtual Machine host and guest” on page 21.

During the planning phase, we made some decisions regarding the IT resources that are needed for our lab environment. This section discusses those decisions.

5.2.1 Configuring resources

For this book, we used the Hardware Manage Console (HMC) and input/output configuration data set (IOCDS) to set up the resources. For more information about IOCDS, see *I/O Configuration Using z/OS HCD and HCM*, [SG24-7804](#).

For users that are unfamiliar with the HMC, the use of Dynamic Partition Manager (DPM) is recommended. For more information, see this [IBM Support web page](#).

5.2.2 Configuring storage resources

In our lab configuration, we decided to use the ECKD DASD configuration as storage devices for the KVM and the guest storages. You can also use SCSI LUNs by using a Fibre Channel Protocol (FCP) configuration, as described in 2.2.2, “Storage considerations” on page 27. On IBM Z, the ECKD disk is accessed through his device address.

After we format the device under Linux, the disk displays a name; for example, starting with dasdx (where x is a - z).

Table 5-1 Storage resources

Device address	Volume name	Capacity	Description
91DE	dasda	400 GB	Rdbkvm1 boot and root disk
904C	dasdb	54 GB	volume group for kvm guest qcow2 files
914C	dasdc	54 GB	volume group for kvm guest qcow2 files

If an FCP SCSI LUNs environment is used, work with your storage team to prepare the disks. The worldwide port name (WWPN) must be provided to the storage team for the correct SAN zoning configuration.

An example of WWPN information that is needed for the zoning is the WWPN of the IBM Z FCP channels and the storage target ports, as shown in the following example:

- ▶ FCP subchannels WWPN:
 - LUN: 4001400800000000
 - FCP: B90A WWPN: C05076D08001DAA8
 - FCP: C90A WWPN: C05076D080009328
- ▶ Storage target PORTS:
 - 5005076309141145 is the WWPN for P1 storage device port
 - 5005076309149145 is the WWPN for P2 storage device port
 - 50050763091b1145 is the WWPN for P3 storage device port
 - 50050763091b9145 is the WWPN for P4 storage device port

Figure 5-3 shows an FCP/SCSI storage area network (SAN) configuration example.

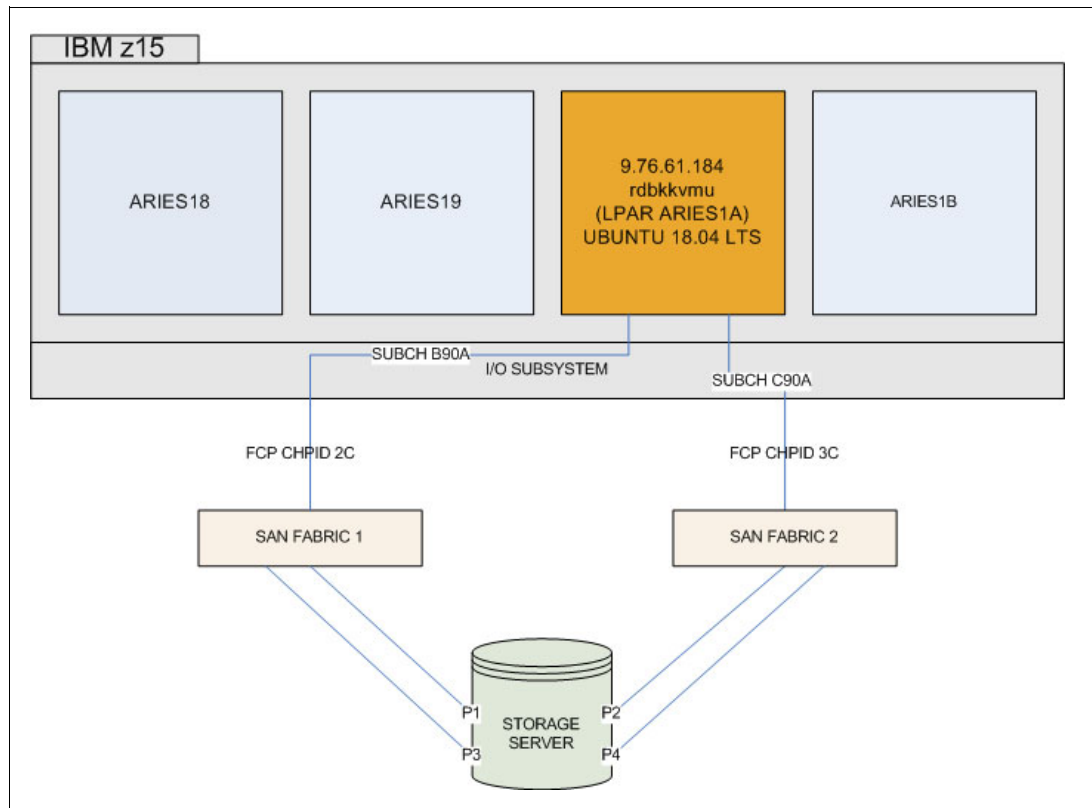


Figure 5-3 Ubuntu SAN configuration

5.2.3 Setting up the FTP server for the installation

In this example, we followed the Ubuntu instructions that are available at this [Ubuntu Wiki web page](#).

You must download the .iso file from Ubuntu 21.10 (Impish Indri).

On our FTP server that included an IP address of rdbkftp1.pbm.ihost.com, we created a directory for each ISO file that was downloaded from Ubuntu images page Ubuntu 21.10 (Impish Indri) and uploaded the contents to the FTP server.

For an Ubuntu 21.10 LTS (impish) installation, the following installation path was used in our FTP server:

```
/var/www/html/ubuntu-21.10-live-server-s390x
```

5.3 Collecting information

Based on the instructions that are provided in the planning stage as described in Chapter 2, “Planning for the Kernel-based Virtual Machine host and guest” on page 21, it is recommended that you save the information that you use during the installation process.

A good practice is to create a table (see Table 5-2) that contains the components’ information. This table is useful during the installation process and for future consultation.

Table 5-2 Sample KVM host installation checklist

Name	Type	Description	More information
Host IP/subnet	TCP/IP	129.40.23.198 / 255.255.255.0	KVM host
	VLAN	8	
Hostname.domain	DNS	rdbkkvm1.pbm.ihost.com	rdbkkvm1.pbm.ihost.com
Gateway	Default GW	129.40.23.254	
FTP server	FTP port 20/21	rdbkftp1.pbm.ihost.com	Check firewall rules
FTP folder	Installation parameter	/var/www/html/ubuntu-21.10-live-server-s390x	Check permission
FTP access	Credentials	User: Inxadmin pw xxx	
LPAR	Logical Partition	Aries38	
Memory	RAIM Memory	128 GB	HostOS, GuestsOS, and Workloads
Physical Processors	IFL (shared)	16 IFL	SMT enabled
Virtual Processors	Virtual Processors	2 for each guest	Can be expanded later, recommended vCPU number <=max of physical CPUs
Storage	ECKD DASD	0.0.91DE 090904C 0.0.914C	400 GB 54 GB 54 GB

Name	Type	Description	More information
Host IP/subnet	TCP/IP	129.40.23.198 / 255.255.255.0	KVM host
	VLAN	8	
Hostname.domain	DNS	rdbkkvm1.pbm.ihost.com	rdbkkvm1.pbm.ihost.com
Gateway	Default GW	129.40.23.254	
FTP server	FTP port 20/21	rdbkftp1.pbm.ihost.com	Check firewall rules
FTP folder	Installation parameter	/var/www/html/ubuntu-21.10-live-server-s390x	Check permission
FTP access	Credentials	User: lnxadmin pw xxx	
LPAR	Logical Partition	Aries38	
Memory	RAIM Memory	128 GB	HostOS, GuestsOS, and Workloads
Physical Processors	IFL (shared)	16 IFL	SMT enabled
OSA1	Network card1	CHP E8	Devices 1E80-1E82
CRYPTO	Domain/Card	CARDS 0x03 0x06	DOMAINS 0x28 0x2A

5.3.1 Required information for Ubuntu on an LPAR installation

In this section, we provide the information about our lab environment. You can use the information in this section as a reference to create your own environment.

Installing by using FTP

The Ubuntu installation points to an FTP server that is provided by Canonical. Be sure to have the FTP port open in the firewall. The FTP server in our lab environment includes the following the pertinent information:

- ▶ IP address: rdbkftp1.pbm.ihost.com
- ▶ Credentials:
 - User lnxadmin
 - Password: xxxxxx
- ▶ Directory: /var/www/html/ubuntu-21.10-live-server-s390x

OSA device addresses

On the IBM Z platform, the Network interfaces (NIC) are represented by OSA express adapters. Each OSA card can manage a range of devices. To use a specific OSA, three consecutive addresses are required: one device for control reads, one for control writes, and the third for data.

For this example, we choose the first triplet from OSA CHPID E8: 1E80-1E82.

Networking information

Contact your network administrator to obtain the networking information for the host.

The following networking information was used in our lab environment:

- ▶ Host name: rdbkkvms
- ▶ IP address: 9.76.61.184
- ▶ Hostname: rdbkkvm1
- ▶ IP address: 129.40.23.198
- ▶ Subnet prefix: 24
- ▶ Default gateway: 129.40.23.254
- ▶ Layer 2 or 3: 2
- ▶ VLAN: 8
- ▶ DNS: 129.40.106.1 and 129.40.106.2

For HiperSockets network access, IP address 100.150.233.70 was used.

Storage

As described in 2.2.2, “Storage considerations” on page 27, two options are available on Linux on z platform: ECKD DASD disk or FCP LUN disk. I

n this example, we used ECKD DASD.

Our example featured the following storage information:

- ▶ ECKD device address: 91DE
- ▶ Volume serial: 0X91DE
- ▶ Space: 400

The operating system installation uses a single DASD under Logical Volume Manager (LVM).

5.3.2 Required information for virtual machine installations

In this section, we review following the required components for VM installations:

- ▶ Compute
- ▶ Memory
- ▶ Disk
- ▶ Network
- ▶ Cryptography

Compute

For VM deployment, all of the guests use two virtual CPUs (vcpu) to use the Simultaneous Multi-Threading (SMT) on an IBM Integrated Facility for Linux (IFL) processor.

Memory

Each VM features 2 GB of RAM, which is the amount of memory that is related to the type of workload that a machine is going to host. For the Linux guest operating system, we recommend starting with 512 MB of memory (see Chapter 2, “Planning for the Kernel-based Virtual Machine host and guest” on page 21).

To avoid memory constraints, it is a good practice to have an accurate workload and capacity study to accurately define the amount of memory.

Disk

QCOW2 is a file format for disk image files that are used by Quick Emulator (QEMU), which is a hosted VM monitor. QEMU Copy On Write uses a disk storage optimization strategy that delays allocation of storage until it is needed. Disk images for specific guest operating systems are often provided as a file in the QCOW2 format.

A QCOW2 image file was used for the operating system disk in our example. The files were stored in the LVM to create more flexible storage migrations.

For more information, see 2.2.2, “Storage considerations” on page 27.

The ECKD DASD used for the Volume Group (VG) that is used for images (rdbkkvm4-images), is the 0X904C volume.

The maximum space that is specified in our lab environment for the image files was 10 GB, but can be extended. We created the following two disk images to use as storage for the virtual machine guests:

- ▶ kvm4guest01: /var/lib/libvirt/images/kvm4guest1_vol001.img
- ▶ kvm4guest02: /var/lib/libvirt/images/kvm4guest2_vol001.img
- ▶ kvm4guest03: /var/lib/libvirt/images/kvm4guest3_vol001.img
- ▶ kvm4guest04: /var/lib/libvirt/images/kvm4guest4_vol001.img

Network

As described in “OSA device addresses” on page 147, you must contact your network team to obtain the proper networking information.

The following guest network setup was used in our lab environment:

- ▶ For external network access:
 - Hostname: kvm4guest01
 - IP address: 129.40.23.212
 - Subnet prefix: 24
 - Default gateway: 129.40.23.254
 - Hostname: kvm4guest02
 - IP address: 129.40.23.213
 - Subnet prefix: 24
 - Default gateway: 129.40.23.254
 - Hostname: kvm4guest03
 - IP address: 129.40.23.214
 - Subnet prefix: 24
 - Default gateway: 129.40.23.254
 - Hostname: kvm4guest04
 - IP address: 129.40.23.215
 - Subnet prefix: 24
 - Default gateway: 129.40.23.254

For HiperSockets access:

- Hostname: rdbkkvm4
- IP address: 100.150.233.60
- Hostname: kvm4guest01
- IP address: 100.150.233.61
- Hostname: kvm4guest02
- IP address: 100.150.233.62
- Hostname: kvm4guest03
- IP address: 100.150.233.63

- Hostname: kvm4guest04
- IP address: 100.150.233.64

Cryptography

For more information about the z15 Crypto Express adapters, see 2.4.5, “Cryptography” on page 42. In our lab environment, we assigned two crypto adapters and three domains to the ARIES38 LPAR.

The Adjunct Processor (AP) queues that we used in our lab environment as our virtual cryptographic resources are listed in Table 5-3.

Table 5-3 AP queues assignment

Crypto domains/Crypto adapters	03 (0x03)	06 (0x6)
40 (0x28)	03.001A	06.001A
41(0x29)	03.004F	06.004F
42(0x2A)	03.0050	06.0050

As described in 2.4.5, “Cryptography” on page 42, the AP queues are a combination of <crypto card>.<crypto domain>., which are expressed in hexadecimal form.

Consider the following points:

- ▶ Domain 40 was used for rdbkvm4
- ▶ Domain 41 was used for kvm4guest1
- ▶ Domain 42 was used for kvm4guest2

5.4 Installing Ubuntu on an LPAR as a KVM host

In this section, we describe how to complete the following tasks:

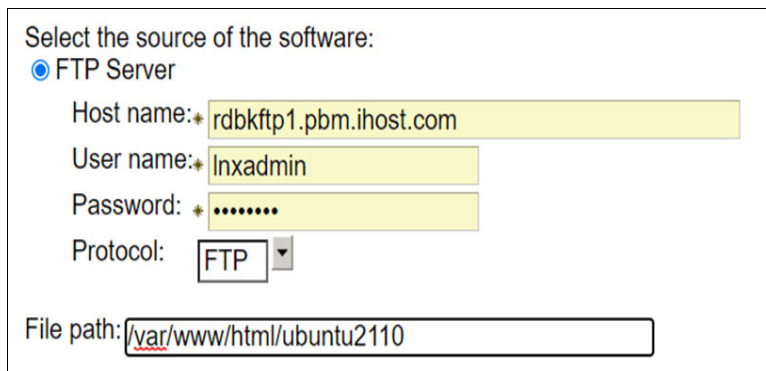
- ▶ Prepare for the installation
- ▶ Install Ubuntu on an LPAR
- ▶ Prepare the host for virtualization

5.4.1 Preparing the installation

For more information about the use of an FTP server to install Ubuntu on an LPAR, see “Installing by using FTP” on page 147.

5.4.2 Installing Ubuntu on an LPAR

After all of the prerequisites were met, we started from FTP by using the information that is described in “Installing by using FTP” on page 147 (see Figure 5-4).



Select the source of the software:

FTP Server

Host name: rdbkftp1.pbm.ihost.com

User name: inxadmin

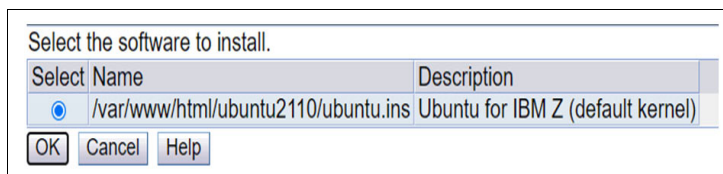
Password:

Protocol: FTP

File path: /var/www/html/ubuntu2110

Figure 5-4 Ubuntu load from removable media or server

In the DPM or HMC, when you receive the prompt with the list of .ins files, choose the file that you created, such as ubuntu.ins (see Figure 5-5).



Select the software to install.

Select	Name	Description
<input checked="" type="radio"/>	/var/www/html/ubuntu2110/ubuntu.ins	Ubuntu for IBM Z (default kernel)

OK Cancel Help

Figure 5-5 Selecting Ubuntu .ins file

Continue with the installation process and use the [Installation In LPAR](#) as guidance.

5.5 Preparing the host for virtualization

Complete the following steps to enable Ubuntu on IBM Z as a KVM Host:

1. Subscribe the server to the Ubuntu network.

For production environments, it is recommended to subscribe the server to the Ubuntu support. For more information, see [this web page](#).

2. Check whether the LPAR supports virtualization functions.

The LPAR must support Start Interpretive Execution (SIE) instructions. Example 5-1 shows how to check SIE support.

Example 5-1 Checking virtualization support

```
root@rdbkvm4:/home/lnxadmin# lscpu | grep sie
Flags:                esan3 zarch stfle msa ldisp eimm dfp edat etf3eh
highgrps te vx vxd vxe gs vxe2 vxp sort dflt sie
```

Example 5-2 shows how to check whether KVM is available and acceleration can be used.

Example 5-2 Checking KVM availability

```
root@rdbkvm4:/home/lnxadmin# kvm-ok

INFO: /dev/kvm exists

KVM acceleration can be used
```

3. Install the virtualization packages and modules.

In this step, you install the virtualization packages by using the commands that are shown in Example 5-3.

Example 5-3 Installing KVM packages

```
root@rdbkvm4:/home/lnxadmin# sudo apt-get install qemu-kvm libvirt-daemon-system
libvirt-clients bridge-utils virtinst
```

Example 5-4 shows how to add the required root user by using the **adduser** command.

Example 5-4 Adding required users

```
root@rdbkvm4:/home/lnxadmin# sudo adduser `id -un` libvirt
Adding user `root' to group `libvirt' ...
Adding user root to group libvirt
Done.
root@rdbkvm4:/home/lnxadmin# sudo adduser `id -un` kvm
Adding user `root' to group `kvm' ...
Adding user root to group kvm
Done.
```

4. Validate whether the host is ready for virtualization.

Before working with KVM, run the **virt-host-validate** command, as shown in Example 5-5.

Example 5-5 Virtualization verification

```
root@rdbkvm4:/home/lnxadmin# virt-host-validate
QEMU: Checking for hardware virtualization                : PASS
QEMU: Checking if device /dev/kvm exists                  : PASS
QEMU: Checking if device /dev/kvm is accessible          : PASS
QEMU: Checking if device /dev/vhost-net exists            : PASS
QEMU: Checking if device /dev/net/tun exists              : PASS
QEMU: Checking for cgroup 'cpu' controller support       : PASS
QEMU: Checking for cgroup 'cpuacct' controller support   : PASS
QEMU: Checking for cgroup 'cpuset' controller support    : PASS
QEMU: Checking for cgroup 'memory' controller support    : PASS
QEMU: Checking for cgroup 'devices' controller support   : PASS
QEMU: Checking for cgroup 'blkio' controller support     : PASS
QEMU: Checking for device assignment IOMMU support       : PASS
QEMU: Checking if IOMMU is enabled by kernel              : PASS
QEMU: Checking for secure guest support                  : WARN
(IBM Secure Execution appears to be disabled in kernel. Add prot_virt=1 to kernel cmdline
arguments)
[...]
```

The warning in the last line of code indicates that this host is not enabled to use secure guest support.

For more information about secure guest support on IBM z15, see 8.3, “Enabling and verifying that the CPC is Secure Execution ready” on page 255.

For more information about planning and implementation, see this [IBM Documentation web page](#).

5.6 Configuring the KVM host

This section describes how to enable Ubuntu as the KVM host and set up the devices to be ready for VM guest usage.

5.6.1 Defining NICs

As described in 5.1, “Defining the target configuration” on page 142, in our lab environment, we used one NIC through the 1e80-1e82 triplet OSA devices (which is defined in the E8 OSA channel) for management purposes. For the VM guest network, we used the MacVTap network that uses a two OSA interfaces (OSA E8 and OSA EE).

As shown in Example 5-6, the only NIC that is configured is the NIC that we used for the Ubuntu installation.

Example 5-6 Configured networks

```
root@rdbkvm4:/home/lnxadmin# znetconf -c
Device IDs                Type      Card Type      CHPID Drv. Name      State
-----
0.0.1e80,0.0.1e81,0.0.1e82 1731/01 OSD_10GIG      E8 qeth enc1e80      online
```

By following the architecture that is proposed in our lab environment for the guest network, we added two NICs (OSA triplets) that use different OSA cards that access the same network through different switches.

Example 5-7 shows two unconfigured NICs that were added with different OSA cards and CHPIDs, which provides redundancy for the virtual environment.

Example 5-7 Checking NICs availability

```
root@rdbkvm4:/home/lnxadmin# znetconf -u | grep 'e8\|ee'
0.0.1e83,0.0.1e84,0.0.1e85 1731/01 OSA (QDIO)      e8 qeth
0.0.1ee3,0.0.1ee4,0.0.1ee5 1731/01 OSA (QDIO)      ee qeth
```

As shown in Example 5-8, we configure the 0.0.1e83-0.0-1e85 device as interface eth0 and the 0.0.1ee3-0.0.0-1ee5 device as interface eth1.

Example 5-8 Configuring the NICs

```
root@rdbkvm4:/home/lnxadmin# chzdev -e qeth 0.0.1e83,0.0.1e84,0.0.1e85 layer2=1
buffer_count=128
QETH device 0.0.1e83:0.0.1e84:0.0.1e85 configured
Note: The initial RAM-disk must be updated for these changes to take effect:
      - QETH device 0.0.1e83:0.0.1e84:0.0.1e85
update-initramfs: Generating /boot/initrd.img-5.13.0-21-generic
```

```
Using config file '/etc/zipl.conf'
Building bootmap in '/boot'
Adding IPL section 'ubuntu' (default)
Preparing boot device: dasda (91de).
Done.
```

```
root@rdbkvm4:/home/lnxadmin#
root@rdbkvm4:/home/lnxadmin#
```

```

root@rdbkvm4:/home/lnxadmin# chzdev -e qeth 0.0.1ee3,0.0.1ee4,0.0.1ee5 layer2=1
buffer_count=128
QETH device 0.0.1ee3:0.0.1ee4:0.0.1ee5 configured
Note: The initial RAM-disk must be updated for these changes to take effect:
    - QETH device 0.0.1ee3:0.0.1ee4:0.0.1ee5
update-initramfs: Generating /boot/initrd.img-5.13.0-21-generic
Using config file '/etc/zipl.conf'
Building bootmap in '/boot'
Adding IPL section 'ubuntu' (default)
Preparing boot device: dasda (91de).
Done.

```

Example 5-9 shows how to check the NICS when the configuration is complete.

Example 5-9 Checking NICS configuration

```

root@rdbkvm4:/home/lnxadmin# lsqeth enc1e83
Device name           : enc1e83
-----
card_type             : OSD_10GIG
cdev0                 : 0.0.1e83
cdev1                 : 0.0.1e84
cdev2                 : 0.0.1e85
chpid                 : E8
online                : 1
portname              : no portname required
portno                : 0
state                 : SOFTSETUP
priority_queueing     : disabled
buffer_count         : 128
layer2                : 1
isolation             : none
bridge_role           : none
bridge_state          : inactive
bridge_hostnotify     : 0
bridge_reflect_promisc : none
switch_attrs          : unknown
vnicc/flooding        : 0
vnicc/learning         : 0
vnicc/learning_timeout : 600
vnicc/mcast_flooding  : 0
vnicc/rx_bcast        : 1
vnicc/takeover_learning : 0
vnicc/takeover_setvmac : 0

```

For more information about network configuration on Ubuntu, see [IBM Documentation](#).

5.6.2 Defining the bond interface

First, check whether the bonding module is enabled. If it is not enabled, enable it and make it persistent after the host is restarted (see Example 5-10).

Example 5-10 Enabling bonding module and make it persistent (adding modules i /etc/modules)

```
root@rdbkvm4:/home/lnxadmin# modprobe bonding

root@rdbkvm4:/home/lnxadmin# cat /etc/modules
# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loaded
# at boot time, one per line. Lines beginning with "#" are ignored.
bonding
```

To enable network high availability (HA), we define a bond interface that is named bond0 (master). This interface accesses the physical network through two NIC slave interfaces: enc1e23 and enc1e43.

Example 5-11 shows how to define a bond interface and set enc1e23 and enc1e43 as slave interfaces on the bond0 interface. To change the properties of these NICs, the interfaces *must* be down.

Example 5-11 Defining bond interface

```
root@rdbkvm4:/home/lnxadmin# ip link add bond0 type bond miimon 100 mode
balance-tlb
root@rdbkvm4:/home/lnxadmin# ip link set enc1e83 down
root@rdbkvm4:/home/lnxadmin# ip link set enc1ee3 down
root@rdbkvm4:/home/lnxadmin# ip link set enc1e83 master bond0
root@rdbkvm4:/home/lnxadmin# ip link set enc1ee3 master bond0
root@rdbkvm4:/home/lnxadmin# ip link set enc1e83 up
root@rdbkvm4:/home/lnxadmin# ip link set enc1ee3 up
root@rdbkvm4:/home/lnxadmin# ip link set bond0 up
```

If you have dedicated OSAs for the slaves and the suitable configuration in the switches, you can configure the bond options as `mode=802.3ad miimon=100`, which allows you to create LACP aggregation groups that share the speed and duplex settings.

If you have two OSA Express7s 10 Gb, you can aggregate two 10 Gb per second (Gbps) ports into a 20 Gbps trunk port. This configuration is equivalent to having one interface with 20 Gbps speed. It also provides fault tolerance and load balancing.

As shown in Example 5-12, we verify that the definition of the bond0 interface is correct.

Example 5-12 Verifying bond interface

```
root@rdbkvm4:/home/lnxadmin# cat /proc/net/bonding/bond0
Ethernet Channel Bonding Driver: v5.13.0-21-generic

Bonding Mode: transmit load balancing
Primary Slave: None
Currently Active Slave: enc1e83
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0
```

Peer Notification Delay (ms): 0

Slave Interface: encle83
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: ea:d1:60:38:33:74
Slave queue ID: 0

Slave Interface: enclee3
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: ee:4c:ae:fc:25:f0
Slave queue ID: 0

Next, we must set the bond0 interface and the slave configuration as permanent in the eth* interfaces.

For this task, we used Netplan. Netplan is a utility that is used to configure network interfaces on Linux. Netplan uses YAML files for configuring network interfaces and is available in the official package repository of Ubuntu.

To install Netplan, update your APT package repository cache; then, run the **sudo apt install netplan** command.

For more information about interface management through Netplan, see [this web page](#).

Example 5-13 shows the section that must be added in the definition file for the bond interface (bond0) in the /etc/netplan/01-netcfg.yaml file.

Example 5-13 Making bond0 and slave interfaces permanent

```
root@rdbkvm4:/etc/netplan# cat 02-netcfg.yaml
network:
  version: 2
  renderer: networkd
  ethernets:
    encle23:
      dhcp4: no
      dhcp6: no
    encle43:
      dhcp4: no
      dhcp6: no
  bonds:
    bond0:
      dhcp4: no
      dhcp6: no
      interfaces:
        - encle23
        - encle43
      parameters:
        mode: balance-tlb
```

Example 5-14 shows the command that is used to apply the network configuration that was created in Example 5-13 on page 157.

Example 5-14 Applying the network configuration

```
root@rdbkvm4:/etc/netplan# netplan apply 02-netcfg.yaml
```

Because we used the VLAN 8 in our lab, we must create a VLAN sub-interface of the bond1 connection (see Example 5-15).

Example 5-15 Creating the bond1.008 VLAN 8 subinterface

```
root@rdbkvm4:/etc/netplan# cat 04-netcfg.yaml
network:
  version: 2
  renderer: networkd
  vlans:
    bond0.8:
      dhcp6: no
      dhcp4: no
      accept-ra: no
      id: 8
      link: bond0
link: bond0
```

Example 5-16 shows the command that is to apply the network configuration that was created in Example 5-15.

Example 5-16 Applying the network configuration

```
root@rdbkvm4:/etc/netplan# netplan apply 04-netcfg.yaml
```

For more information about bonding, see the IBM publication [Linux Channel Bonding Best Practices and Recommendations](#).

5.6.3 Defining HiperSockets interfaces

HiperSockets allows memory-to-memory communication between hosts in the same IBM Z platform. HiperSockets avoids the use of external communications by way of a NIC and Ethernet switch, which eliminates traditional network latency.

For more information about this feature, see “Network connectivity” on page 6.

As described in 5.1, “Defining the target configuration” on page 142, the HiperSocket CHPID is F4 and triplet for the encf00 interface definition is 0F00-0F02 in our lab environment.

5.6.4 Defining HiperSocket interface to support VM guest network

We define the encf00 interface on the HiperSocket chipid (f4) to allow VM guests access to the HiperSocket network.

Example 5-17 shows the HiperSocket device availability.

Example 5-17 List of unconfigured HSI devices on F4 CHPID

```
root@rdbkvm4:/etc/netplan# znetconf -u | grep " f4 "
```

0.0.0f00,0.0.0f01,0.0.0f02	1731/05	HiperSockets	f4	qeth
0.0.0f03,0.0.0f04,0.0.0f05	1731/05	HiperSockets	f4	qeth
0.0.0f06,0.0.0f07,0.0.0f08	1731/05	HiperSockets	f4	qeth
0.0.0f09,0.0.0f0a,0.0.0f0b	1731/05	HiperSockets	f4	qeth
0.0.0f0c,0.0.0f0d,0.0.0f0e	1731/05	HiperSockets	f4	qeth
0.0.0f0f,0.0.0f10,0.0.0f11	1731/05	HiperSockets	f4	qeth
0.0.0f12,0.0.0f13,0.0.0f14	1731/05	HiperSockets	f4	qeth
0.0.0f15,0.0.0f16,0.0.0f17	1731/05	HiperSockets	f4	qeth
0.0.0f18,0.0.0f19,0.0.0f1a	1731/05	HiperSockets	f4	qeth
0.0.0f1b,0.0.0f1c,0.0.0f1d	1731/05	HiperSockets	f4	qeth

Choose the 0.0.0f00,0.0.0f01,0.0.0f02 devices to create the encf00 interface, as shown in Example 5-18.

Example 5-18 Configuring the HiperSocket interface and verifying the assigned name

```
root@rdbkvm4:/etc/netplan# chzdev -e qeth 0.0.0f00,0.0.0f01,0.0.0f02 layer2=1
buffer_count=128
QETH device 0.0.0f00:0.0.0f01:0.0.0f02 configured
root@rdbkvm4:/etc/netplan# lsqeth encf00
Device name          : encf00
```

```
-----
card_type            : HiperSockets
cdev0                : 0.0.0f00
cdev1                : 0.0.0f01
cdev2                : 0.0.0f02
chpid                : F4
online               : 1
portname             : no portname required
portno               : 0
state                : SOFTSETUP
priority_queueing    : disabled
buffer_count         : 128
layer2               : 1
isolation             : none
bridge_role          : none
bridge_state         : inactive
bridge_hostnotify    : 0
bridge_reflect_promisc : none
vnicc/bridge_invisible : 0
vnicc/flooding       : 0
vnicc/learning       : 0
vnicc/learning_timeout : 600
vnicc/mcast_flooding : 0
vnicc/rx_bcast       : 1
vnicc/takeover_learning : 0
vnicc/takeover_setvmac : 0
```

You also can define a HiperSockets interface for KVM use. To define this interface, select the 0.0.0f03,0.0.0f04,0.0.0f05 devices to create interface encf03, as shown in Example 5-19.

Example 5-19 Configuring the HiperSocket interface.

```
root@rdbkvm4:/etc/netplan# chzdev -e qeth 0.0.0f03,0.0.0f04,0.0.0f05 layer2=1
buffer_count=128
QETH device 0.0.0f03:0.0.0f04:0.0.0f05 configured
root@rdbkvm4:/etc/netplan# lsqeth encf03
Device name           : encf03
-----
      card_type       : HiperSockets
      cdev0           : 0.0.0f03
      cdev1           : 0.0.0f04
      cdev2           : 0.0.0f05
      chpid           : F4
      online          : 1
      portname        : no portname required
      portno          : 0
      state           : SOFTSETUP
      priority_queueing : disabled
      buffer_count    : 128
      layer2          : 1
      isolation       : none
      bridge_role     : none
      bridge_state    : inactive
      bridge_hostnotify : 0
      bridge_reflect_promisc : none
      vnicc/bridge_invisible : 0
      vnicc/flooding  : 0
      vnicc/learning  : 0
      vnicc/learning_timeout : 600
      vnicc/mcast_flooding : 0
      vnicc/rx_bcast  : 1
      vnicc/takeover_learning : 0
      vnicc/takeover_setvmac : 0
```

Assign the IP address to the interface and start the interface, as shown in Example 5-20.

Example 5-20 Assigning IP and start encf03 interface

```
root@rdbkvm4:/home/lxadmin# ip link set encf03 up
root@rdbkvm4:/home/lxadmin# ip a add 100.150.233.70/24 dev encf03
```

Example 5-13 shows the section that must be added to the definition file for encf03 interface in the /etc/netplan/01-netcfg.yaml file in ethernets: section.

Example 5-21 Making encf03 interface configuration permanent

```
root@rdbkvm4:/etc/netplan# cat 01-netcfg.yaml
[...]
encf03:
  addresses: [ 100.150.233.70/24 ]
[...]
```

5.6.5 Define HiperSocket Converged Interface

By using HiperSockets Converged Interface (HSCI) connections, a HiperSockets network interface can be combined with an external OSA- or RoCE port, which creates a single network interface. By using this interface, the switched network and the intra-CEC HiperSocket network can be accessed by using the same IP. Both of the devices that participate in the HSCI interface must have the same physical network (PNET) ID (see Table 5-4).

Table 5-4 Lab adapters

Device type	CHPID	Devices	PNETID
HiperSockets	F2	0.0.0FC9, 0.0.0FCA, 0.0.0FCB	PERFNET
OSA Express	EE	0.0.01EE9, 0.0.01EEB, 0.0.01EEA	PERFNET

Example 5-22 - Example 5-27 on page 162 show the steps that are required to define the HiperSocket Converged Interface.

Example 5-22 Checking the PNET ID of the HiperSocket device

```
root@rdbkvm4:/home/lxadmin/isos# cat /sys/devices/css0/chp0.f2/util_string |  
iconv -f IBM-1047 -t ASCII  
PERFNET
```

Example 5-23 Checking the OSA PNET ID

```
root@rdbkvm4:/home/lxadmin/isos# cat /sys/devices/css0/chp0.ee/util_string |  
iconv -f IBM-1047 -t ASCII  
PERFNET
```

Example 5-24 Creating HSI and OSA Interfaces

```
root@rdbkvm4:/home/lxadmin# chzdev -e qeth 0.0.0fc9,0.0.0fca,0.0.0fcb layer2=1  
buffer_count=12  
QETH device 0.0.0fc9:0.0.0fca:0.0.0fcb configured  
Note: The initial RAM-disk must be updated for these changes to take effect:  
- QETH device 0.0.0fc9:0.0.0fca:0.0.0fcb  
update-initramfs: Generating /boot/initrd.img-5.13.0-22-generic  
Using config file '/etc/zipl.conf'  
Building bootmap in '/boot'  
Adding IPL section 'ubuntu' (default)  
Preparing boot device: dasda (91de).  
Done.  
root@rdbkvm4:/home/lxadmin# chzdev -e qeth 0.0.1ee9,0.0.1eea,0.0.1eeb layer2=1  
buffer_count=12  
QETH device 0.0.1ee9:0.0.1eea:0.0.1eeb configured  
Note: The initial RAM-disk must be updated for these changes to take effect:  
- QETH device 0.0.1ee9:0.0.1eea:0.0.1eeb  
update-initramfs: Generating /boot/initrd.img-5.13.0-22-generic  
Using config file '/etc/zipl.conf'  
Building bootmap in '/boot'  
Adding IPL section 'ubuntu' (default)  
Preparing boot device: dasda (91de).  
Done.
```

Example 5-25 Creating the HCSI Interface

```
root@rdbkvm4:/home/lnxadmin# hsci add encfc9 enclee9
Verifying net dev enclee9 and HiperSockets dev encfc9
Adding hsci0fc9 with a HiperSockets dev encfc9 and an external dev enclee9
Set encfc9 MAC 0e:00:f2:38:00:0b on enclee9 and hsci0fc9
Successfully added HSCI interface hsci0fc9
```

Example 5-26 Creating the VLAN 8 interface from hsci 0FC9 device and assign IP

```
root@rdbkvm4:/home/lnxadmin# ip link add dev hsci0fc9.8 link hsci0fc9 type vln id 8
root@rdbkvm4:/home/lnxadmin# ip addr add 129.40.23.231/24 dev hsci0fc9.8
root@rdbkvm4:/home/lnxadmin# ip link set up hsci0fc9.8
```

Example 5-27 Checking HSCI interface

```
root@rdbkvm4:/home/lnxadmin/isos# hsci show
HSCI      PNET_ID      HiperSockets      External
-----
hsci0fc9  PERFNET      encfc9             enclee9
```

5.6.6 Defining SMC interfaces

SMC-D and SMC-R use shared memory to provide low-latency, high-bandwidth, cross-LPAR connections for applications. This support is intended to provide application-transparent direct memory access (DMA) communications to TCP endpoints for socket-based connections.

Installing SMC tools package

To support SMC-D (ISM) and SMC-R (RoCE) you must install the SMC tools package. For more information about obtaining the packages, see this [GitHub web page](#).

Use the commands that are shown in Example 5-28.

Example 5-28 Installing the SMC-tools package

```
root@rdbkvm4:/home/lnxadmin/isos# apt-get install smc-tools
```

Then, unpack the smc-tools that were downloaded and compile the libraries (see Example 5-29 and Example 5-29).

Example 5-29 Installing compiler tools

```
root@rdbkvm4:/home# apt-get install gcc
root@rdbkvm4:/home# sudo apt-get install libnl-genl-3-dev
```

Example 5-30 Compiling smc objects

```
root@rdbkvm4:/home/lnxadmin/isos/smc-tools-main# make
```

Enabling SMC-D

In this section, we provide the basic commands to enable SMC-D on the Ubuntu host server.

Example 5-31 shows how to check the ISM device availability.

Example 5-31 Checking PCI devices

```
root@rdbkvm4:/home/lnxadmin/isos# lspci
0000:00:00.0 Non-VGA unclassified device: IBM Internal Shared Memory (ISM) virtual PCI
device
0001:00:00.0 Non-VGA unclassified device: IBM Internal Shared Memory (ISM) virtual PCI
device
0002:00:00.0 Non-VGA unclassified device: IBM Internal Shared Memory (ISM) virtual PCI
device
0003:00:00.0 Ethernet controller: Mellanox Technologies MT27710 Family [ConnectX-4 Lx
Virtual Function]
0004:00:00.0 Ethernet controller: Mellanox Technologies MT27710 Family [ConnectX-4 Lx
Virtual Function]
0005:00:00.0 Ethernet controller: Mellanox Technologies MT27710 Family [ConnectX-4 Lx
Virtual Function]
0006:00:00.0 Ethernet controller: Mellanox Technologies MT27710 Family [ConnectX-4 Lx
Virtual Function]
```

As shown in Example 5-32 and Example 5-33, we check the PNET ID in the ISM device and in the OSA, which should display the same PNET ID.

Example 5-32 Checking PNET ID of the ISM device

```
root@rdbkvm4:/home/lnxadmin/isos# cat
/sys/devices/pci0000:00/0000:00:00.0/util_string | iconv -f IBM-1047 -t ASCII
PERFNET
```

Example 5-33 Checking the OSA PNET ID

```
root@rdbkvm4:/home/lnxadmin/isos# cat /sys/devices/css0/chp0.ee/util_string |
iconv -f IBM-1047 -t ASCII
PERFNET
```

In our lab, we define a NIC in CHPID EE by using the command that is shown in Example 5-34. For more information, see 5.6.1, “Defining NICs” on page 154.

Example 5-34 Defining OSA and assigning IP

```
root@rdbkvm4:/home/lnxadmin/smc-tools# root@rdbkvm4:/home/lnxadmin/isos# chzdev
-e qeth 0.0.1ee6,0.0.1ee7,0.0.1ee8 layer2=1
QETH device 0.0.1ee6:0.0.1ee7:0.0.1ee8 configured
root@rdbkvm4:/home/lnxadmin/smc-tools# ip an add 129.40.23.224/24 dev encl1ee6
```

To test the communication between rdbkkvm4 and rdbkkvm2 LPARs in the same CPC by using the SMC-D, we use the iperf3 tool. To install it, run the yum command in each LPAR that is shown in Example 5-35.

Example 5-35 Installing iperf3 tool

```
root@rdbkkvm4:/etc/netplan# apt-get install iperf3
[root@rdbkkvm2 home]# yum -y install iperf3
```

Allow the local firewall to accept connections for iperf3 on the 5201 TCP port on rdbkkvm2 LPAR server (see Example 5-36).

Example 5-36 Allowing firewall port 5201

```
[root@rdbkkvm2 home]# firewall-cmd --permanent --add-port=5201/tcp
success
[root@rdbkkvm2 home]# firewall-cmd --reload
success
[root@rdbkkvm2 ~]# firewall-cmd --list-all
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: bond1 bond1.008 enP6p0s0 enP6p0s0.008 enc1e80 enc1e80.008 enc1e83
enc1ee3 enc1ee6 enc1ee6.008 encf00 encf03
  sources:
  services: cockpit dhcpv6-client ssh
  ports: 21/tcp 5901/tcp 5201/tcp
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

Start iperf3 in listening mode by using the command on rdbkkvm2 that is shown in Example 5-37.

Example 5-37 Starting iperf3 using SMC

```
[root@rdbkkvm2 ~]# smc_run iperf3 -s
-----
Server listening on 5201
-----
```

Use the command that is shown in Example 5-38 to open another SSH session against the rdbkkvm2 server and print the SMC sockets information.

Example 5-38 Checking SMC listening on port 5201

```
[root@rdbkkvm2 ~]# smcss -a
State      UID   Inode  Local Address          Peer Address          Intf
Mode
LISTEN     00000 0526700 0.0.0.0:5201
```

To test SMC connections, issue the `iperf3` command on `rdbkvm4` that is shown in Example 5-39 and check on `rdbkvm2`.

Example 5-39 Running iperf3 client on rdbkvm4 to the server on rdbkvm2

```

root@rdbkvm4:/home/lnxadmin# smc_run iperf3 -c 129.40.23.221 -t 10
Connecting to host 129.40.23.221, port 5201
[ 5] local 129.40.23.198 port 45722 connected to 129.40.23.221 port 5201
[ ID] Interval          Transfer      Bitrate      Retr  Cwnd
[ 5]  0.00-1.00    sec  4.13 GBytes  35.5 Gbits/sec    0  14.1 KBytes
[ 5]  1.00-2.00    sec  4.12 GBytes  35.4 Gbits/sec    0  14.1 KBytes
[ 5]  2.00-3.00    sec  4.13 GBytes  35.4 Gbits/sec    0  14.1 KBytes
[ 5]  3.00-4.00    sec  3.87 GBytes  33.2 Gbits/sec    0  14.1 KBytes
[ 5]  4.00-5.00    sec  3.92 GBytes  33.7 Gbits/sec    0  14.1 KBytes
[ 5]  5.00-6.00    sec  4.03 GBytes  34.6 Gbits/sec    0  14.1 KBytes
[ 5]  6.00-7.00    sec  4.04 GBytes  34.7 Gbits/sec    0  14.1 KBytes
[ 5]  7.00-8.00    sec  4.10 GBytes  35.2 Gbits/sec    0  14.1 KBytes
[ 5]  8.00-9.00    sec  4.08 GBytes  35.1 Gbits/sec    0  14.1 KBytes
[ 5]  9.00-10.00   sec  4.09 GBytes  35.2 Gbits/sec    0  14.1 KBytes
- - - - -
[ ID] Interval          Transfer      Bitrate      Retr
[ 5]  0.00-10.00   sec  40.5 GBytes  34.8 Gbits/sec    0
[ 5]  0.00-10.00   sec  40.5 GBytes  34.8 Gbits/sec    0
                                     sender
                                     receiver

iperf Done.

```

Example 5-40 shows the command that is used to test the use of the SMC-D.

Example 5-40 Checking the use of SMC-D

```

[root@rdbkvm2 ~]# smcss -a
State      UID  Inode  Local Address          Peer Address          Intf
Mode
ACTIVE     00000 0326142 ::ffff:129.40.23...:5201 ::ffff:129.40.2...:45722 0000
SMCD
ACTIVE     00000 0326141 ::ffff:129.40.23...:5201 ::ffff:129.40.2...:45718 0000
SMCD
LISTEN     00000 0326139 0.0.0.0:5201
[root@rdbkvm2 ~]# smcss -D
State      UID  Inode  Local Address          Peer Address          Intf
Mode  GID          Token          Peer-GID          Peer-Token          Linkid
ACTIVE     00000 0326142 ::ffff:129.40.23...:5201 ::ffff:129.40.2...:45722 0000
SMCD 12000facb7f88561 0000090e10000000 06000faeb7f88561 0000090f10000000 00000400
ACTIVE     00000 0326141 ::ffff:129.40.23...:5201 ::ffff:129.40.2...:45718 0000
SMCD 12000facb7f88561 0000091010000000 06000faeb7f88561 0000091110000000 00000400

```

If you receive `LD_PRELOAD_error`, run the commands that are shown in Example 5-41.

Example 5-41 LD_PRELOAD_error

```

root@rdbkvm4:/home# smc_run iperf3 -c 129.40.23.221 -t 10
ERROR: ld.so: object 'libsmc-preload.so' from LD_PRELOAD cannot be preloaded
(cannot open shared object file): ignored.

```

You must create a symbolic link in `/usr/lib` for the shared object file `libsmc-preload.so` (see Example 5-42).

Example 5-42 Creating a symbolic link for the shared object file

```
root@rdbkvm4:/home# cd /usr/lib
root@rdbkvm4:/usr/lib# sudo ln -s
/home/lnxadmin/isos/smc-tools-main/libsmc-preload.so libsmc-preload.so
```

You can use de ISM if multiple subnets are used in your configuration on the same CEC.

Note: Direct Memory Access (SMC-D) is enhanced to remove the same subnet restriction by using SMC-Dv2.

SMC-R

As described in “Enabling SMC-D” on page 163, SMC also can be enabled between different CPCs by using a RoCE card that allows remote direct memory access (RDMA) over the external network (SMC-R).

Example 5-43 shows how to check the RoCE device availability.

Example 5-43 Checking PCI devices

```
root@rdbkvm4:/home# lspci
0006:00:00.0 Ethernet controller: Mellanox Technologies MT27710 Family [ConnectX-4
Lx Virtual Function]
```

In Example 5-33 on page 163, the PNET ID in the OSA card is displayed. Example 5-44 shows the PNET ID in the RoCE device, which should display the same PNET ID.

Example 5-44 Checking RoCE device PNET ID

```
root@rdbkvm4:/home# cat /sys/devices/pci0006:00/0006:00:00.0/util_string | iconv
-f IBM-1047 -t ASCII
```

PERFNET

We use the same configuration as in the SMC-D configuration. However, we use RoCE 2 instead ISM in this case.

Example 5-45 shows a similar example that is shown in Example 5-40. However, the communication uses SMCR here.

Example 5-45 Test results

```
[root@rdbkvm2 ~]# smcss -a
State      UID   Inode  Local Address          Peer Address          Intf
Mode
ACTIVE     00000 0338408 ::ffff:129.40.23...:5201 ::ffff:129.40.2...:60256 0000
SMCR
ACTIVE     00000 0338407 ::ffff:129.40.23...:5201 ::ffff:129.40.2...:60254 0000
SMCR
LISTEN     00000 0338405 0.0.0.0:5201
[root@rdbkvm2 ~]# smcss -R
State      UID   Inode  Local Address          Peer Address          Intf
Mode Role IB-device      Port Linkid  GID
Peer-GID
```

```
ACTIVE          00000 0338408 ::ffff:129.40.23...:5201 ::ffff:129.40.2...:60256 0000
SMCR SERV m1x5_0          01 01 0000:0000:0000:0000:ffff:8128:17df
0000:0000:0000:0000:0000:ffff:8128:17e1
ACTIVE          00000 0338407 ::ffff:129.40.23...:5201 ::ffff:129.40.2...:60254 0000
SMCR SERV m1x5_0          01 01 0000:0000:0000:0000:ffff:8128:17df
0000:0000:0000:0000:0000:ffff:8128:17e1
```

If multiple subnets exist between IBM z15 CECS, SMC Version 2 (SMCv2) enables multiple IP subnet capability for SMC. The multiple subnet capability is enabled by updates to the underlying networking specifications for RoCE (referred to as *RoCEv2*) and the IBM Z Internal Shared Memory (ISM) feature (referred to as *ISMv2*) along with updates to the related technologies.

For more information about RoCE, see this [IBM Documentation web page](#).

5.6.7 Defining the MacVTap network

This section describes the definition of two MacVTap networks: one for OSA and another for HiperSockets.

MacVTap for an OSA NIC

Instead of the use of the default network connectivity for the guests network address translation (NAT) connections, we chose MacVTap in bridge mode. This mode allows the guests a direct connection with the specified interface in the MacVTap network.

To configure the MacVTap network, we use the `virsh` command and an XML definition file.

Example 5-46 shows our `macvtap-net.xml` network definition file.

Example 5-46 macvtap-net.xml

```
root@rdbkvm4:/home/lnxadmin/isos# cat macvtap-net1.xml
<network>
  <name>macvtap-net1</name>
  <forward mode="bridge">
    <interface dev="bond0.8"/>
  </forward>
</network>
```

Example 5-47 shows the `virsh` command that is used to define a MacVTap network.

Example 5-47 virsh net-define, net-start, and net-autostart commands

```
root@rdbkvm4:/home/lnxadmin/isos# virsh net-define macvtap-net1.xml
Network macvtap-net1 defined from macvtap-net1.xml
root@rdbkvm4:/home/lnxadmin/isos# virsh net-start macvtap-net1
Network macvtap-net1 started
root@rdbkvm4:/home/lnxadmin/isos# virsh net-autostart macvtap-net1
Network macvtap-net1 marked as autostarted
```

MacVTap for HiperSockets NIC

Example 5-48 shows the XML file that is created to define the HiperSockets NIC.

Example 5-48 macvtap-hsi.xml

```
[root@rdbkvm4 images]# cat macvtap-hsi0.xml
<network>
  <name>macvtap-hsi0</name>
  <forward mode="bridge">
    <interface dev="hsi0"/>
  </forward>
</network>
```

Also define, start, and autostart the new macvtap-hsi1 network, as shown in Example 5-47 on page 167 and Example 5-48.

5.6.8 Defining crypto adapters and domain

As described in 2.2.4, “Encryption considerations” on page 33, the Crypto Express card advantages can be used by the KVM hosts and VM guests.

It is important to check the compatibility list for Crypto Express adapters when Ubuntu is used before beginning the installation. For more information about supported Crypto Express adapters with your version of Ubuntu, see this [IBM Documentation web page](#).

To make the AP cards available to the KVM guests (see 2.4.5, “Cryptography” on page 42), use the VFIO mediated device framework to assign cryptographic adapter resources to the device.

For more information, see *Configuring Crypto Express Adapters for KVM Guests*, [SC34-7717](#).

In our lab environment, we assigned two crypto adapters and one domain to the ARIES38 LPAR (see “Cryptography” on page 150). Use the VFIO mediated device framework to assign cryptographic card resources to the device.

To make this assignment, load the `vfio_ap` device driver by running the commands that are shown in Example 5-49 - Example 5-51 on page 169.

Example 5-49 Enabling vfio_ap permanently (adding modules in /etc/modules)

```
root@rdbkvm4:/home/lxadmin# cat /etc/modules
# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loaded
# at boot time, one per line. Lines beginning with "#" are ignored.
vfio_ap
```

Example 5-50 Preparing Crypto usage

```
root@rdbkvm4:/home/lxadmin/isos# modprobe vfio_ap
root@rdbkvm4:/home/lxadmin/isos# lsmod | grep vfio_ap
vfio_ap                28672  0
mdev                   28672  3 vfio_ccw,vfio_mdev,vfio_ap
vfio                   45056  4 vfio_ccw,vfio_mdev,vfio_iommu_type1,vfio_ap
```

Example 5-51 Freeing all cards and domains from the KVM host

```
root@rdbkvm4:/home/lnxadmin/isos# echo 0x0 > /sys/bus/ap/apmask
root@rdbkvm4:/home/lnxadmin/isos# echo 0x0 > /sys/bus/ap/aqmask
```

Use the `lszcrypt` command to display information about the crypto adapters, as shown in Example 5-52.

Example 5-52 Verifying crypto cards

```
root@rdbkvm4:/home/lnxadmin# lszcrypt
CARD.DOMAIN TYPE  MODE      STATUS  REQUESTS
-----
03           CEX7C CCA-Coproc  online    1
06           CEX7C CCA-Coproc  online    0
```

Assign AP queues to KVM. Example 5-53 shows the procedure to assign the two crypto cards (03 and 06) and domain (0x28) to the KVM host.

Example 5-53 Crypto for KVM host

```
root@rdbkvm4:/home/lnxadmin# echo +0x03 > /sys/bus/ap/apmask
root@rdbkvm4:/home/lnxadmin# echo +0x06 > /sys/bus/ap/apmask
root@rdbkvm4:/home/lnxadmin# echo +0x28 > /sys/bus/ap/aqmask
```

Example 5-54 shows the verification of the crypto assignment to the KVM host.

Example 5-54 Verifying crypto assignment

```
root@rdbkvm4:/home/lnxadmin# lszcrypt
CARD.DOMAIN TYPE  MODE      STATUS  REQUESTS
-----
03           CEX7C CCA-Coproc  online    2
03.0028      CEX7C CCA-Coproc  online    2
06           CEX7C CCA-Coproc  online    0
06.0028      CEX7C CCA-Coproc  online    0
```

One way to make the configuration of the cryptos permanent is by running scripts and using `rc.local` to run the scripts at start. Example 5-55 shows how to enable `rc.local` systemd to run scripts.

Example 5-55 Checking whether rc_local service is running

```
root@rdbkvm4:/home/lnxadmin# sudo systemctl status rc-local
? rc-local.service - /etc/rc.local Compatibility
   Loaded: loaded (/lib/systemd/system/rc-local.service; static)
   Drop-In: /usr/lib/systemd/system/rc-local.service.d
           ??debian.conf
   Active: inactive (dead)
     Docs: man:systemd-rc-local-generator(8)
root@rdbkvm4:/home/lnxadmin# sudo systemctl enable rc-local
The unit files have no installation config (WantedBy=, RequiredBy=, Also=,
Alias= settings in the [Install] section, and DefaultInstance= for template
units). This means they are not meant to be enabled using systemctl.
```

Example 5-56 - Example 5-60 on page 171 shows the sequence of commands used to create and make the crypto configuration permanent.

Example 5-56 Creating the rc_local service systemd

```
root@rdbkvm4:/home/lnxadmin# sudo nano /etc/systemd/system/rc-local.service
root@rdbkvm4:/home/lnxadmin# cat /etc/systemd/system/rc-local.service
[Unit]
Description=/etc/rc.local Compatibility
ConditionPathExists=/etc/rc.local

[Service]
Type=forking
ExecStart=/etc/rc.local start
TimeoutSec=0
StandardOutput=tty
RemainAfterExit=yes
SysVStartPriority=99

[Install]
WantedBy=multi-user.target
```

Example 5-57 Creating the rc_local file

```
root@rdbkvm4:/home/lnxadmin# sudo nano /etc/rc.local
root@rdbkvm4:/home/lnxadmin# cat /etc/rc.local
#!/bin/sh -e
##
exit 0
root@rdbkvm4:/home/lnxadmin# sudo chmod -v +x /etc/rc.local
```

Example 5-58 Enabling and starting the rc_local service

```
root@rdbkvm4:/home/lnxadmin# sudo systemctl enable rc-local
Created symlink /etc/systemd/system/multi-user.target.wants/rc-local.service ?
/etc/systemd/system/rc-local.service.
root@rdbkvm4:/home/lnxadmin# sudo systemctl enable rc-local
Created symlink /etc/systemd/system/multi-user.target.wants/rc-local.service ?
/etc/systemd/system/rc-local.service.
root@rdbkvm4:/home/lnxadmin# sudo systemctl start rc-local.service
root@rdbkvm4:/home/lnxadmin# sudo systemctl status rc-local.service
• rc-local.service - /etc/rc.local Compatibility
  Loaded: loaded (/etc/systemd/system/rc-local.service; enabled; vendor preset:
enabled)
  Drop-In: /usr/lib/systemd/system/rc-local.service.d
          --debian.conf
  Active: active (exited) since Tue 2021-11-30 15:08:45 UTC; 3s ago
  Process: 1465 ExecStart=/etc/rc.local start (code=exited, status=0/SUCCESS)
  CPU: 1ms
```

Example 5-59 Creating script for freeing the ap and aq queues for crypto enablement

```
root@rdbkvm4:/home/lnxadmin/isos# vim crypto_enablement.sh
root@rdbkvm4:/home/lnxadmin/isos# chmod u+x
/home/lnxadmin/isos/crypto_enablement.sh
root@rdbkvm4:/home/lnxadmin/isos# cat crypto_enablement.sh
#!/bin/bash
# Freeing ap and aq queues for crypto enablement
echo Preparing crypto enviroment
echo 0x0 > /sys/bus/ap/apmask
echo 0x0 > /sys/bus/ap/aqmask
echo +0x03 > /sys/bus/ap/apmask
echo +0x06 > /sys/bus/ap/apmask
echo +0x28 > /sys/bus/ap/aqmask
```

Example 5-60 Adding the script to rc_local

```
root@rdbkvm4:/home/lnxadmin# sudo chmod u+x
/home/lnxadmin/isos/crypto_enablement.sh
root@rdbkvm4:/home/lnxadmin# sudo nano /etc/rc.local
root@rdbkvm4:/home/lnxadmin/isos# cat /etc/rc.local
#!/bin/sh -e
##
/home/lnxadmin/isos/crypto_enablement.sh
exit 0
```

The guest uses the crypto adapter and domains through mediated devices. How to create a mediated device for `vfio_ap` is shown in Example 5-61.

Example 5-61 Defining mediated device

```
<hostdev mode='subsystem' type='mdev' managed='no' model='vfio-ap'>
  <source>
    <address uuid='e67ad324-50ad-4321-bba5-ee235ca0e45a' />
  </source>
</hostdev>
```

Example 5-62 shows how to generate an Universally Unique Identifier (UUID) for the mediated device, create the mediated device, and assign the crypto cards and crypto domains to it (for use and control).

Example 5-62 Generating an UUID for VM guest

```
root@rdbkvm4:/home/lnxadmin/isos# uuidgen
e67ad324-50ad-4321-bba5-ee235ca0e45a
root@rdbkvm4:/home/lnxadmin# echo e67ad324-50ad-4321-bba5-ee235ca0e45a
> /sys/devices/vfio_ap/matrix/mdev_supported_types/vfio_ap-passthrough/create
root@rdbkvm4:/home/lnxadmin/isos# echo 0x03 >
/sys/devices/vfio_ap/matrix/e67ad324-50ad-4321-bba5-ee235ca0e45a/assign_adapter
root@rdbkvm4:/home/lnxadmin/isos# echo 0x06 >
/sys/devices/vfio_ap/matrix/e67ad324-50ad-4321-bba5-ee235ca0e45a/assign_adapter
root@rdbkvm4:/home/lnxadmin/isos# echo 0x0029 >
/sys/devices/vfio_ap/matrix/e67ad324-50ad-4321-bba5-ee235ca0e45a/assign_domain
root@rdbkvm4:/home/lnxadmin/isos# echo 0x0029 >
/sys/devices/vfio_ap/matrix/e67ad324-50ad-4321-bba5-ee235ca0e45a/assign_control_domain
```

The procedure that is shown in Example 5-63 must be done for each domain that is used by a VM. In our lab environment, we used domains 41 and 42. Example 5-63 also shows how to verify the mediated device crypto assignment.

Example 5-63 Verifying mediated device crypto assignment

```
root@rdbkvm4:/home/lnxadmin/isos# cat
/sys/devices/vfio_ap/matrix/e67ad324-50ad-4321-bba5-ee235ca0e45a/matrix
03.0029
06.0029
```

To make the `vfio_ap` persistent, you must install the `mdevctl` package and perform the commands that are shown in Example 5-64 by using the UUID, adapter, and domains that are used for the mediated device.

Example 5-64 Making vfio_ap mediated device persistent

```
root@rdbkvm4:/home/lnxadmin# mdevctl define --uuid
e67ad324-50ad-4321-bba5-ee235ca0e45a --parent matrix --type vfio_ap-passthrough
root@rdbkvm4:/home/lnxadmin# mdevctl modify --uuid
e67ad324-50ad-4321-bba5-ee235ca0e45a --addattr=assign_adapter --value=0x03
root@rdbkvm4:/home/lnxadmin# mdevctl modify --uuid
e67ad324-50ad-4321-bba5-ee235ca0e45a --addattr=assign_adapter --value=0x06
root@rdbkvm4:/home/lnxadmin# mdevctl modify --uuid
e67ad324-50ad-4321-bba5-ee235ca0e45a --addattr=assign_domain --value=0x0029
root@rdbkvm4:/home/lnxadmin# mdevctl modify --uuid
e67ad324-50ad-4321-bba5-ee235ca0e45a --addattr=assign_control_domain
--value=0x0029
root@rdbkvm4:/home/lnxadmin# mdevctl start --uuid
e67ad324-50ad-4321-bba5-ee235ca0e45a
root@rdbkvm4:/home/lnxadmin# mdevctl modify --uuid
e67ad324-50ad-4321-bba5-ee235ca0e45a --auto
root@rdbkvm4:/home/lnxadmin# mdevctl list
ecded9cd-6fa4-483c-90c0-62203c3d0b11 matrix vfio_ap-passthrough (defined)
```

Notes: Consider the following points:

- ▶ The mediated device *must* be started after the host is restarted.
- ▶ The `--auto` parameter that is used in the Example 5-64 on page 172 triggers only autostart after a system restart if the user sets the `apmask` and `aqmask` correctly *and* loads the `vfio_ap` driver.

5.7 Deploying virtual machines on KVM

In this section, we describe the deployment of VMs in the KVM environment. Although a VM can be created by using several methods, this section describes the use of the `virt-install` command and `virsh` tools.

5.7.1 Creating QCOW2 disk image file

As described in “Disk” on page 149, QCOW2 files are used to create the VM disks.

Example 5-65 shows the command that is used to create a QCOW2 file of 10 GB.

Example 5-65 Creating QCOW2 image file

```
root@rdbkvm4:/home/lnxadmin/isos# qemu-img create -f qcow2 kvm4guest01_vol001.img 10G
Formatting 'kvm4guest01_vol001.img', fmt=qcow2 cluster_size=65536 extended_l2=off
compression_type=zlib size=10737418240 lazy_refcounts=off refcount_bits=16
```

5.7.2 Installing a new guest by using virt-install

The `virt-install` command-line tool is used for creating VMs on KVM, which uses the `libvirt` hypervisor management library.

Example 5-66 shows how to install a VM by using the `virt-install` command.

Example 5-66 Creating VM guest using virt-install command

```
root@rdbkvm4:/home# virt-install --name kvm4guest01 --memory 4000 --vcpus 2
--os-variant ubuntu20.10 --import --disk
path=/home/lnxadmin/isos/kvm4guest01_vol001.img --network network:macvtap-net1
--cdrom /home/lnxadmin/isos/ubuntu-21.10-live-server-s390x.iso
```

Consider the following points:

- ▶ The `--name` parameter specifies the name of the VM guest.
- ▶ The `--memory` parameter specifies the amount of memory (RAM) that is allocated to the virtual machine (expressed in megabytes).
- ▶ The `--vcpus` parameter specifies how many vcpus are assigned to the VM.
- ▶ The `--disk` parameter specifies the media to use as storage for the VM guest (kvm4guest01 uses QCOW2 files). If the file was preallocated, specify the `--import` parameter. Otherwise, you can omit the `--import` parameter and use the new file path by using the parameters' format and size to allocate the file during the installation.
- ▶ The `--os-variant` parameter specifies which type of operating system is to be installed; this option is highly recommended when importing a disk image. If it is not provided, the performance of the created VM is negatively affected. Run the `osinfo-query os` command to see a full list of available operating systems.
- ▶ The `--network` parameter specifies the network options for the VM guest. In this case, we are connecting the guest to the MacVTap network.
- ▶ For the installation source, we used a `.iso` file that uses the `--cdrom` parameter. You can also install from other sources, such as an FTP server.

After the command is issued (see Example 5-66 on page 173), the VM installation begins, as shown in Figure 5-6.

```
Starting install...
Running text console command: virsh --connect qemu:///system console
kvm4guest01
Connected to domain 'kvm4guest01'
Escape character is ^] (Ctrl + ])
[ 0.082933] Linux version 5.13.0-19-generic (buildd@bos02-s390x-014) (gcc
(Ubuntu 11.2.0-7ubuntu2) 11.2.0, GNU ld (GNU Binutils for Ubuntu) 2.37)
#19-Ubuntu SMP Thu Oct 7 20:16:26 UTC 2021 (Ubuntu 5.13.0-19.19-generic
5.13.14)
[ 0.082937] setup: Linux is running under KVM in 64-bit mode
[ 0.084744] setup: The maximum memory size is 4000MB
[ 0.084792] cpu: 2 configured CPUs, 0 standby CPUs
[ 0.084871] Write protected kernel read-only data: 16988k
[...]
[ 4.741581] loop5: detected capacity change from 0 to 488016
[ 4.778775] overlayfs: "xino" feature enabled using 32 upper inode bits.
done.
[ 5.052217] systemd[1]: systemd 248.3-1ubuntu8 running in system mode. (+PAM
+AUDIT +SELINUX +APPARMOR +IMA +SMACK +SECCOMP +GCRYPT +GNUTLS -OPENSSL +ACL
+BLKID +CURL +ELFUTILS -FIDO2 +IDN2 -IDN +IPTC +KMOD +LIBCRYPTSETUP -LIBFDISK
+PCRE2 -PWQUALITY -P11KIT -QRENCODE +BZIP2 +LZ4 +XZ +ZLIB +ZSTD -XKBCOMMON
+UTMP +SYSVINIT default-hierarchy=unified)
[ 5.052464] systemd[1]: Detected virtualization kvm.
[ 5.052467] systemd[1]: Detected architecture s390x.

Welcome to Ubuntu 21.10!

[ 5.062870] systemd[1]: Initializing machine ID from random generator.
[ 5.414334] systemd[837]: /usr/lib/systemd/system-generators/s390-cpi-vars
failed with exit status 1.
[ 5.470611] systemd[1]: cdrom.mount: Unit is bound to inactive unit
dev-sr0.device. Stopping, too.
[ 5.471169] systemd[1]: Queued start job for default target Graphical
Interface.
```

Figure 5-6 VM guest installation process through `virt-install`

For more information about the `virt-install` command, see this [Ubuntu wiki web page](#).

5.7.3 Cloning a guest using Virsh

The **virsh** command-line tool is used to manage VM guests and the hypervisor. It also uses the **libvirt** hypervisor management library. In this section, we show how to clone a VM from a previous image installation base.

Example 5-67 shows the first task; that is, copy the QCOW2 file `kvmuvm01_vol001.img` to `kvmuvm02_vol001.img`.

Example 5-67 Copying the QCOW2 file

```
root@rdbkvm4:/var/lib/libvirt/images# cp kvm4gues01_vol001.img >
kvm4guest02_vol001.img
```

Use the **dumpxml** command to return the guest VM machine's configuration file. As shown in Example 5-68, we obtain the XML configuration file, `kvmuvm02.xml`, from the VM guest, `kvmuvm01`.

Example 5-68 Dumping kvmuvm01 guest definition file

```
root@rdbkvm4:/var/lib/libvirt/images# virsh dumpxml kvmu4guest01 >
kvm4guest02.xml
```

Because we are going to clone this VM guest, edit `kvm4gues02.xml` and make the following changes:

- ▶ Change the VM name from:
`<name>kvmsvm01</name>`
to:
`<name>kvmsvm02</name>`
- ▶ Delete the following UUID assignment statement:
`<uuid>d370d9de-a881-45b7-80a5-94d09b447d15</uuid>`
- ▶ Change the source file of QCOW2 disk from:
`<source file='/home/lnxadmin/isos/kvm4guest01_vol001.img' />`
to:
`<source file='/home/lnxadmin/isos/kvm4guest02_vol001.img' />`
- ▶ In the `<interface type='direct'>` section, delete the following statements:
 - MAC address: `<mac address='52:54:00:57:3a:69' />`
 - Target devices statement: `<target dev='macvtap3' />`

All deleted information is dynamically generated when the **virsh define** command is used.

The `kvm4guest02` guest is defined, as shown in Example 5-69.

Example 5-69 kvm4guest02 guest definition

```
root@rdbkvm4:/home# virsh define kvm4guest02.xml
Domain kvm4guest02 defined from kvm4guest02.xml
```

To start the new cloned guest, run a **virsh start kvm4guest02** command.

You must change the basic parameters of the new guest, such as the IP address and host name.

Another way for cloning guest is by using the `virt-clone` command, as shown in Example 5-70. The guest *must* be shut down to be cloned.

Example 5-70 Cloning the kvm4guest01 guest

```
root@rdbkvm4:/home/lnxadmin/isos# virt-clone --original kvm4guest01 --name
kvm4guest02 --file /home/lnxadmin/isos/kvm4guest02_vol001.img
Allocating 'kvm4guest02_vol001.img'
| 10 GB 00:00:00
```

```
Clone 'kvm4guest02' created successfully.
```

Consider the following points:

- ▶ The `--original` statement indicates the name of the guest (domain) to be cloned.
- ▶ The `--name` statement indicates the name of the new guest (domain) to be created.
- ▶ The `--file` statement indicates the location of the new qcow2 to be allocated for the new guest.

5.7.4 Adding HiperSockets to the VM guest

To add an NIC, a VM is needed to shut down the guest and edit the domain definition. In this case, we use a vNIC, `macvtap-hsi`, which targets the `encf00` HiperSocket interface.

Example 5-71 shows the command that is used to edit the VM domain definition in XML format.

Example 5-71 Editing domain definition

```
root@rdbkvm4:/home/lnxadmin# virsh edit kvm4guest01
Domain kvm4guest01 XML configuration edited.
```

You must also add the definition that is shown in Example in the `<devices>` `</devices>` section.

Example 5-72 Interface definition

```
<devices>
<interface type='network'>
    <source network='macvtap-hsi1'/>
    <model type='virtio'/>
</interface>
</devices>
```

After the domain starts, the VM shows the new interface and that the domain definition was updated (see Example 5-73).

Example 5-73 Interface verification

At the VM level:

```
root@kvm4guest01:/etc/netplan# ip a show enc6
3: enc6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
default qlen 1000
    link/ether 52:54:00:dc:96:d9 brd ff:ff:ff:ff:ff:ff
    inet 100.150.233.61/24 brd 100.150.233.255 scope global enc6
        valid_lft forever preferred_lft forever
    inet6 fe80::5054:ff:fedc:96d9/64 scope link
        valid_lft forever preferred_lft forever
```

At the KVM host:

```
root@rdbkvm4:/home/lnxadmin/isos# virsh dumpxml kvm4guest01
[...]
  <interface type='direct'>
    <mac address='52:54:00:dc:96:d9' />
    <source network='macvtap-hsi0' portid='562cf85e-269c-47ed-9719-47bad088d4ab'
dev='encf00' mode='bridge' />
    <target dev='macvtap5' />
    <model type='virtio' />
    <alias name='net1' />
    <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0006' />
  </interface>
[...]
```

5.7.5 Adding space to guest from ECKD DASD

To add space to a VM in our lab, we added a full volume DASD as a virtio device. You also can add space by using a logical volume from an LVM pool.

The first step is formatting the volume on the host (see Example 5-74) shows for the ECKD device 0.0.914C. For this task, we must check whether the device is available for the LPAR and enabled it before the formatting process.

Example 5-74 DASD formatting

```
root@rdbkvm4:/home/lnxadmin/isos# chzdev -e dasd 0.0.914C
ECKD DASD 0.0.914c configured
Note: The initial RAM-disk must be updated for these changes to take effect:
    - ECKD DASD 0.0.914c
update-initramfs: Generating /boot/initrd.img-5.13.0-22-generic
Using config file '/etc/zipl.conf'
Building bootmap in '/boot'
Adding IPL section 'ubuntu' (default)
Preparing boot device: dasda (91de).
Done.
root@rdbkvm4:/home/lnxadmin/isos# dasdfmt -b 4096 /dev/disk/by-path/ccw-0.0.914c
-p --label=0x914C
Drive Geometry: 60102 Cylinders * 15 Heads = 901530 Tracks
Device Type: Thinly Provisioned
```

I am going to format the device `/dev/disk/by-path/ccw-0.0.914c` in the following way:

```
Device number of device : 0x914c
  Labelling device       : yes
  Disk label             : VOL1
  Disk identifier        : 0X914C
  Extent start (trk no)  : 0
  Extent end (trk no)    : 1
  Compatible Disk Layout : yes
  Blocksize              : 4096
  Mode                   : Quick
  Full Space Release     : yes
```

WARNING:

Disk `/dev/disk/by-path/ccw-0.0.914c` is online on operating system instances in 15 different LPARs.

Ensure that the disk is not being used by a system outside your LPAR.

Note: Your installation might include z/VM systems that are configured to automatically vary on disks, regardless of whether they are subsequently used.

---> ATTENTION! <---

All data of that device will be lost.

Type "yes" to continue, no will leave the disk untouched: yes

Releasing space for the entire device...

Skipping format check due to thin-provisioned device.

Formatting the first two tracks of the device.

Finished formatting the device.

Rereading the partition table... ok

To obtain more I/O performance on the virtual block devices, we can configure one or more I/O threads for the virtual server and each virtual block device can use one of these I/O threads (see Example 5-75).

Example 5-75 Creating I/O thread for guest

```
root@rdbkvm4:/home/lxadmin/isos# virsh iothreadadd --domain kvm4guest01 --id 1
--live
root@rdbkvm4:/home/lxadmin/isos# virsh iothreadadd --domain kvm4guest01 --id 1
--config
```

Now, we can define and attach to the guest the new formatted DASD. Remember to always format the DASD on the host, but create the partitions at the guest level if you plan to assign the entire disk to the guest (see Example 5-76).

Example 5-76 Defining the virtual block device .xml and attach to the guest

```
root@rdbkvm4:/home/lxadmin/isos# vim kvm4guest01_dasd01.xml
root@rdbkvm4:/home/lxadmin/isos# cat kvm4guest01_dasd01.xml
<disk type="block" device="disk">
  <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
  <source dev="/dev/disk/by-path/ccw-0.0.914c"/>
  <target dev="vdb" bus="virtio"/>
  <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x00a8"/>
</disk>
```

```

root@rdbkvm4:/home/lnxadmin/isos# virsh attach-device kvm4guest01
kvm4guest01_dasd01.xml --persistent
Device attached successfully

```

Example 5-77 shows the KVM and guest commands that are used to verify weather the virtual block device was successfully created.

Example 5-77 Verification commands

In KVM, we can verify the usage of the virtio device:

```

root@rdbkvm4:/home/lnxadmin/isos# virsh domblklist kvm4guest01
Target  Source
-----
vda     /home/lnxadmin/isos/kvm4guest01_vo1001.img
vdb     /dev/disk/by-path/ccw-0.0.914c
sda     -

```

On the guest, we verify the device availability:

```

root@kvm4guest01:/home/rdbkuser1# lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
loop0                               7:0      0  53.3M 1 loop /snap/core20/1167
loop1                               7:1      0  53.3M 1 loop /snap/core20/1240
loop2                               7:2      0  39.2M 1 loop /snap/snapd/14057
loop3                               7:3      0  30.2M 1 loop /snap/snapd/13266
loop4                               7:4      0   65M 1 loop /snap/lxd/21620
loop5                               7:5      0  64.7M 1 loop /snap/lxd/21898
sr0                                 11:0     1 1024M 0 rom
vda                                 252:0    0   10G 0 disk
--vda1                             252:1    0    1G 0 part /boot
--vda2                             252:2    0    9G 0 part
  ??ubuntu--vg-ubuntu--lv         253:0    0    9G 0 lvm /
vdb                                 252:16   0  41.3G 0 disk

```

5.7.6 Adding DASD space to a guest as a VFIO device

Another way to add DASD to the guest is by using a VFIO pass-through device, which allows the guest to control the hole DASD as a direct device. To add the DASD to the guest, you must bring its subchannel under control of the `vfio_ccw` device driver, create a mediated device for the DASD and then, assign the mediated device to the guest (see Example 5-78 - Example 5-84 on page 181).

Example 5-78 Checking the device subchannel

```

root@rdbkvm4:/home/lnxadmin# lscss -a | grep 904c
Device  Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
-----
0.0.904c 0.0.24ea  3390/0c 3990/e9    f0 f0 ff  40424143 00000000

```

Example 5-79 Formatting the DASD device

```

root@rdbkvm4:/home/lnxadmin# dasdfmt -b 4096 /dev/disk/by-path/ccw-0.0.904c -p
--label=0x904C

```

Example 5-80 Unbinding the device from host and creating the mediated device for DASD

```
root@rdbkvm4:/home/lnxadmin# echo 0.0.904C >
/sys/bus/ccw/drivers/dasd-eckd/unbind
root@rdbkvm4:/home/lnxadmin# echo 0.0.24ea >
/sys/bus/css/devices/0.0.24ea/driver/unbind
root@rdbkvm4:/home/lnxadmin# echo 0.0.24ea > /sys/bus/css/drivers/vfio_ccw/bind
root@rdbkvm4:/home/lnxadmin# uuidgen
4d56cc2f-f614-4549-934d-656eef9926f5
root@rdbkvm4:/home/lnxadmin# echo 4d56cc2f-f614-4549-934d-656eef9926f5>
/sys/bus/css/devices/0.0.24ea/mdev_supported_types/vfio_ccw-io/create
```

Example 5-81 Adding the mediated device to the definition in the device section

```
[root@rdbkvm4 by-path]# virsh edit kvm1guest01
<hostdev mode="subsystem" type="mdev" model="vfio-ccw">
  <source>
    <address uuid="4d56cc2f-f614-4549-934d-656eef9926f5"/>
  </source>
  <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x00b1"/>
</hostdev>
```

Example 5-82 Checking at the guest level

```
root@kvm4guest01:/home/rdbkuser1# lscss
Device  Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
-----
0.0.0002 0.0.0000  0000/00 3832/08 yes  80  80  ff  00000000 00000000
0.0.0003 0.0.0001  0000/00 3832/03 yes  80  80  ff  00000000 00000000
0.0.0000 0.0.0002  0000/00 3832/02 yes  80  80  ff  00000000 00000000
0.0.0001 0.0.0003  0000/00 3832/01 yes  80  80  ff  00000000 00000000
0.0.0006 0.0.0004  0000/00 3832/01 yes  80  80  ff  00000000 00000000
0.0.0004 0.0.0005  0000/00 3832/05 yes  80  80  ff  00000000 00000000
0.0.00b1 0.0.0008  3390/0c 3990/e9   f0  f0  7f  50525153 00000000
0.0.0005 0.0.0006  0000/00 3832/04 yes  80  80  ff  00000000 00000000
0.0.00a8 0.0.0007  0000/00 3832/02 yes  80  80  ff  00000000 00000000
```

Example 5-83 Enabling the DASD on the guest

```
root@kvm4guest01:/home/rdbkuser1# chzdev -e dasd 0.0.00b1
ECKD DASD 0.0.00b1 configured
Note: The initial RAM-disk must be updated for these changes to take effect:
  - ECKD DASD 0.0.00b1
update-initramfs: Generating /boot/initrd.img-5.13.0-22-generic
Using config file '/etc/zipl.conf'
Building bootmap in '/boot'
Adding IPL section 'ubuntu' (default)
Preparing boot device: vda (0000).
Done.
```

Example 5-84 Verifying the device availability at guest level

```
root@kvm4guest01:/home/rdbkuser1# lsdasd
Bus-ID   Status   Name     Device  Type           BlkSz  Size      Blocks
-----
0.0.00b1 active   dasda    94:0    ECKD (ESE)     4096   42259MB   10818360
root@kvm1guest1:/home/rdbkuser1# lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
loop1                               7:1    0  53.3M  1 loop /snap/core20/1240
loop2                               7:2    0  39.2M  1 loop /snap/snapd/14057
loop3                               7:3    0   65M   1 loop /snap/lxd/21620
loop4                               7:4    0  30.2M  1 loop /snap/snapd/13639
loop5                               7:5    0  64.7M  1 loop /snap/lxd/21898
loop6                               7:6    0  53.4M  1 loop /snap/core20/1272
sr0                                  11:0    1 1024M  0 rom
dasda                                94:0    0  41.3G  0 disk
  > dasda1                          94:1    0  41.3G  0 part
    active dasda 94:0    0  41.3G  0 disk
root@kvm4guest01:/home/rdbkuser1# lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0                                  11:0    1 1024M  0 rom
dasda                                94:0    0  41.3G  0 disk
  > dasda1                          94:1    0  41.3G  0 part
```

To make the `vfiocccw` persistent, you must install the `mdevctl` and `driverctl` packages and then, run the commands that are shown in Example 5-85 by using the UUID and the subchannel selected for this device.

Notes: Consider the following points:

- ▶ Mediated devices can be configured manually by using `sysfs` operations.
- ▶ The `mdevctl` utility is used for managing and persisting devices in the mediated device framework of the Linux kernel. For more information, see this [GitHub web page](#).
- ▶ The `driverctl` device driver is a control utility for Linux. For more information, see this [GitLab web page](#).

Example 5-85 Making vfiocccw mediated device persistent

```
root@rdbkvm4:/home/lxadmin# mdevctl define -u
fef02213-6b37-4434-9158-c4105c8d2b6f -p 0.0.24ea -t vfiocccw-io
root@rdbkvm4:/home/lxadmin# mdevctl start --uuid
fef02213-6b37-4434-9158-c4105c8d2b6f
root@rdbkvm4:/home/lxadmin# mdevctl modify --uuid
fef02213-6b37-4434-9158-c4105c8d2b6f --auto
root@rdbkvm4:/home/lxadmin# mdevctl list
fef02213-6b37-4434-9158-c4105c8d2b6f 0.0.24ea vfiocccw-io (defined)
```

5.7.7 Adding LUNs if you have FCP Storage

To add space to a VM, we must map the target LUN. In this case, we choose an available LUN to identify the device ID that we enable in the VM.

As described in 2.2.2, “Storage considerations” on page 27, the following options are available:

- ▶ Entire disk (LUN or ECKD DASD)
- ▶ Partition of the disk or
- ▶ A logical volume

It is important to map the device ID by using the multipath ID. In some installations, this mapping can be achieved by using multipath-friendly names such as `mpathX`. To be read by VM migrations, the recommendation is avoid the use of multipath-friendly names.

Example 5-86 shows how to identify the target LUN.

Example 5-86 LUN identification

```
root@rdbkvm4:/home/lxadmin# multipath -ll | grep
36005076309ffd14500000000000010a
mpathi (36005076309ffd14500000000000010a) dm-7 IBM,2107900
```

Example 5-87 shows the identification by device ID.

Example 5-87 Device mapper mpath identification by device ID

```
root@rdbkvm4:/dev/disk/by-id# ls | grep 36005076309ffd14500000000000010a
dm-uuid-mpath-36005076309ffd14500000000000010a
scsi-36005076309ffd14500000000000010a
```

After identifying the target LUN and the device ID for our lab environment, the target disk is shown in the following example:

```
/dev/disk/by-id/dm-uuid-mpath-36005076309ffd14500000000000010a
```

You must assign a target device in the domain. You also must check which devices are being used. Example 5-88 shows how to list the used devices in the domain.

Example 5-88 Device list example

```
root@rdbkvm4:/var/lib/libvirt/images# virsh domblklist kvmvm01
Target Source
-----
vda      /var/lib/libvirt/images/kvmvm01_vol001.img
```

With this information available, the next step is to create an XML file to attach the disk that is free. The commands that are used in Example 5-89 show that the vdb device is available.

Example 5-89 Device mapper mpath identification by device ID

```
root@rdbkvm4:/var/lib/libvirt/images# vim kvmvm01_block1.xml
root@rdbkvm4:/var/lib/libvirt/images# cat kvmvm01_block1.xml
  <disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native"/>
    <source
dev="/dev/disk/by-id/dm-uuid-mpath-36005076309ffd14500000000000010a"/>
<target dev='vdb' bus='virtio'/>
  </disk>
```

Define the disk to the VM guest, as shown in Example 5-90.

Example 5-90 Attaching disk on kvm4guest01 guest

```
root@rdbkvm4:/var/lib/libvirt/images# virsh attach-device kvm4guest01
kvmvm01_block1.xml --persistent
Device attached successfully
```

Validate the host and the guest, as shown in Example 5-91.

Example 5-91 Attaching disk verification

From KVM host:

```
root@rdbkvm4:/var/lib/libvirt/images# virsh domblklist kvmvm01
Target  Source
-----
vda     /var/lib/libvirt/images/kvm4guest01_vol001.img
vdb     /dev/disk/by-id/dm-uuid-mpath-36005076309ffd14500000000000010a
```

From kvm4guest01 guest:

```
root@kvm4guest01:/home/lnxadmin# lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda                                  252:0    0   10G  0 disk
├─vda1                               252:1    0   10G  0 part
│   ├─kvmvm01--vg-root              253:0    0    9G  0 lvm  /
│   └─kvmvm01--vg-swap_1            253:1    0   976M  0 lvm  [SWAP]
vdb                                  252:16   0    40G  0 disk
```

5.7.8 Adding cryptography support to the VM guest

In 5.6.8, “Defining crypto adapters and domain” on page 168, the crypto adapters and domain were defined. The AP queues were then assigned for use by KVM. The `vfiop` mediated device was created to enable the assignment of the crypto device to a VM guest.

Complete the following steps to add cryptography support to the VM guest:

1. In the VM domain definition, edit the XML file (see Example 5-92).

Example 5-92 Editing VM definitions using `virsh`

```
root@rdbkvm4:/var/lib/libvirt/images# virsh edit kvm4guest01
Domain kvm4guest01 XML configuration edited.
```

2. Locate the `<devices>` section and add the `<hostdev>` section, as shown in Example 5-93.

Example 5-93 Mediated device definition

```
<hostdev mode='subsystem' type='mdev' managed='no' model='vfiop'>
  <source>
    <address uuid='e67ad324-50ad-4321-bba5-ee235ca0e45a' />
  </source>
</hostdev>
```

The true random number generator (TRNG) feature can be used to generate random numbers. You can enable this feature as shown in Example 5-94. For more information, see Chapter 2, “Planning for the Kernel-based Virtual Machine host and guest” on page 21.

Example 5-94 Statement to use TRNG

```
<rng model='virtio'>
  <backend model='random'>/dev/trng</backend>
</rng>
```

When you start the guest, you might see the message that is shown in the Example 5-95.

Example 5-95 Error in `/dev/trng`

```
root@rdbkvm4:/home/lnxadmin/isos# virsh start kvm4guest01 --console
error: Failed to start domain kvm4guest01
error: internal error: process exited while connecting to monitor:
2021-11-29T14:33:05.316521Z qemu-system-s390x: -object
rng-random,id=objrng0,filename=/dev/trng: Could not open '/dev/trng': Permission
denied
```

A `Permission denied` message can occur because this specific device does not have read or write permission. To check the condition, run the command that is shown in Example 5-96.

Example 5-96 Error in `/dev/trng`

```
root@rdbkvm4:/home/lnxadmin/isos# dmesg | grep /dev/trng
[ 4310.601112] audit: type=1400 audit(1638225131.437:75): apparmor="DENIED"
operation="open" profile="libvirt-f27ac39f-0833-43d1-ab06-e5618bb0981d"
name="/dev/trng" pid=12908 comm="qemu-system-s390" requested_mask="r"
denied_mask="r" fsuid=64055 ouid=0
```

Correct this condition by adding “/dev/trng rw,” in the file /etc/apparmor.d/abstractions/libvirt-qemu. Reload the apparmor service by using the **service apparmor reload** command.

For more information, see [Ubuntu AppArmor](#).

3. Recycle the VM and verify the definitions by running the commands that are shown in Example 5-97.

Example 5-97 Verification commands

In KVM we verify the usage of TRNG:

```
root@rdbkvm4:/home/lnxadmin/isos# cat /sys/devices/virtual/misc/trng/byte_counter
trng: 16
hwrng: 16
arch: 1920
total: 1952
```

On the guest we verify the crypto availability:

```
root@kvm4guest01:/home/rdbkuser1# lscrypt
CARD.DOMAIN TYPE MODE STATUS REQUESTS
-----
03          CEX7C CCA-Coproc online      1
03.0029     CEX7C CCA-Coproc online      1
06          CEX7C CCA-Coproc online      0
06.0029     CEX7C CCA-Coproc online      0
```

Upon completion of these steps, the crypto card is available to be used in the entire environment, including the KVM host and the VMs.

For more information, see *Linux on Z and LinuxONE: Configuring Crypto Express Adapters for KVM Guests*, [SC34-7717](#).

5.7.9 Using the Integrated Accelerator for zEnterprise Data Compression

The Integrated Accelerator for zEnterprise Data Compression (zEDC) with the IBM z15 replaces the zEDC Express adapter with on-chip compression, which provides increased throughput and capacity. It also reduces the cost of storing, processing, and transporting data.

The acceleration with the on-chip Integrated Accelerator for zEDC is available to applications that use `zlib` or `gzip` in user space and to the kernel `zlib`.

To check whether your platform can use the Integrated zEDC, you must check whether you have the `dflt` feature available by using the command that is shown in Example 5-98.

Example 5-98 Checking dflt feature

```
kvm4guest01:/home/lnxadmin # lscpu | grep dflt
Flags:                    esan3 zarch stfle msa ldisp eimm dfp edat etf3eh
highgrps te vx vxd vxe gs vxe2 vxp sort dflt
```

The `df1t` feature is available in our lab because we use a z15. To use this feature, we must update the `DFLTCC_LEVEL_MASK` environmental variable that establishes the compression level.

You can update this environmental variable for a command, a session, or the entire system. For this example, we show the variable update for a session level by using a session level that uses a 5.8 Gb text file that is called `operlog.txt`.

By using higher levels of compression, you can save almost 30x more CPU time and 23x more elapsed time; however, the ratio compression can be lower (in this case, only 2.7%)

Table 5-5 lists the compression statistics.

Table 5-5 Compression statistics

Compression level	Compression ratio	Elapsed time	Total CPU time
0x0000	91.6%	1m 24.713s	1m 23641s
0x0002	91.65%	1m 24.212s	1m 14.829s
0x007E	88.9%	0m 3.441s	0m 2.848s
0x01FF	88.9%	0m 4.376s	0m 2.760s

Figure 5-7 shows reports of compression exercises.

```

root@kvm4guest01:/home# rm -rf operlog1_nohw.gz
root@kvm4guest01:/home# export DFLTCC_LEVEL_MASK=0x0000
root@kvm4guest01:/home# time gzip -v -c operlog.txt > operlog1_nohw.gz
operlog.txt:  91.6%
real  1m24.713s
user  1m21.485s
sys   0m2.156s

root@kvm4guest01:/home# export DFLTCC_LEVEL_MASK=0x0002
root@kvm4guest01:/home# time gzip -v -c operlog.txt > operlog1_lvl_0x0002.gz
operlog.txt:  91.6%
real  1m24.212s
user  1m21.683s
sys   0m2.146s

root@kvm1guest1:/home/rdbkuser1# export DFLTCC_LEVEL_MASK=0x007e
root@kvm4guest01:/home# export DFLTCC_LEVEL_MASK=0x007e
root@kvm4guest01:/home# time gzip -v -c operlog.txt > operlog1_lvl_0x007e.gz
operlog.txt:  88.9%
real  0m3.441s
user  0m0.447s
sys   0m2.401s

root@kvm4guest01:/home# time gzip -v -c operlog.txt > operlog1_lvl_0x01ff.gz
operlog.txt:  88.9%
real  0m4.376s
user  0m0.445s
sys   0m2.315s

```

Figure 5-7 Compression exercises



Managing the Kernel-based Virtual Machine environment

After the Kernel-based Virtual Machine (KVM) environment is running, other tasks must be done, such as changing the environment, recovering data, and keeping the environment secure.

This chapter reviews various tools that can be used in the KVM environment for the main enterprise distributions for the following management domains:

- ▶ Resources management: The process of assigning real and virtual resources to different entities and making configuration changes to virtual machines (VMs).
- ▶ Recovery management: An approach for backing up data and running data recovery in a timely and reliable fashion.
- ▶ Security management: A practice that ensures authorized access to data, systems, and resources is secure and that an audit trail exists if any violations occur.

This chapter does *not* include High Availability because most of those concepts are not unique to KVM and can be used in other virtualization environments.

This chapter includes the following topics:

- ▶ 6.1, “Managing resources” on page 188
- ▶ 6.2, “Recovery management” on page 197
- ▶ 6.3, “Security management” on page 201

6.1 Managing resources

A virtualized environment consists of real and virtual resources, such as CPUs, memory, network interfaces, networks, and storage. The purpose of a resource management tool is to allow efficient workflows for defining, assigning, modifying, and removing such resources.

This section covers the following open source resource management tools:

- ▶ Virsh
- ▶ Virt-manager
- ▶ Cockpit
- ▶ OpenStack

6.1.1 Virsh

Virsh is the main command line interface (CLI) of `libvirt` for managing VMs and other resources. Typically, virsh uses XML as the definition language for the VMS, networks, and other resources.

Note: The virsh CLI can be installed as follows for the different distributions:

- ▶ RHEL: `yum install libvirt`
- ▶ Ubuntu: `apt-get install libvirt-bin`
- ▶ SUSE: `zypper install libvirt`

Example 6-1 shows part of a VM definition (domain) in the XML file (content suppressed).

Example 6-1 Virsh domain definition

```
[root@rdbkvm2 /]# virsh dumpxml guestVM1
<domain type='kvm' id='11'>
  <name>guestVM1</name>
  <uuid>9e2bc4ee-77d6-4928-8639-83317da0eb64</uuid>
  <metadata>
    <libosinfo:libosinfo
      xmlns:libosinfo="http://libosinfo.org/xmlns/libvirt/domain/1.0">
      <libosinfo:os id="http://ubuntu.com/ubuntu/20.10"/>
    </libosinfo:libosinfo>
  </metadata>
  <domain type='kvm' id='10'>
    </cpu>
    <clock offset='utc' />
    <on_poweroff>destroy</on_poweroff>
    <on_reboot>restart</on_reboot>
    <on_crash>destroy</on_crash>
  [...]
</domain>
```

Example 6-2 and Example 6-3 show other useful commands for virsh.

Example 6-2 Listing host information by using virsh

```
[root@rdbkvm2]# sudo virsh nodeinfo
CPU model:          s390x
CPU(s):             16
CPU frequency:     5200 MHz
CPU socket(s):     1
Core(s) per socket: 16
Thread(s) per core: 2
NUMA cell(s):      1
Memory size:       263046652 KiB
```

Example 6-3 Virsh maintenance commands

Get Domain List

```
[root@rdbkvm2 /]# virsh list --all
 Id   Name      State
-----
 11   guestVM1  running
```

Get domain detail info

```
[root@rdbkvm2 /]# virsh dominfo guestVM1
setlocale: No such file or directory
Id:          11
Name:        guestVM1
UUID:        9e2bc4ee-77d6-4928-8639-83317da0eb64
OS Type:     hvm
State:       running
CPU(s):      2
CPU time:    19.8s
Max memory:  4194304 KiB
Used memory: 4194304 KiB
Persistent:  yes
Autostart:   enable
Managed save: no
Security model: selinux
Security DOI: 0
Security label: system_u:system_r:svirt_t:s0:c121,c280 (enforcing)
```

Get virtual network list

```
[root@rdbkvm2 /]# virsh net-list
 Name      State  Autostart  Persistent
-----
 default   active yes         yes
 macvtap-net1 active yes         yes
 OpenNet   active no          yes
```

```
[root@rdbkvm2 /]# virsh vol-list default
```

```
 Name      Path
-----
 guestVM1-datavol /var/lib/libvirt/images/guestVM1-datavol
 GuestVM1-Vol1    /var/lib/libvirt/images/GuestVM1-Vol1
 GuestVM1-vol1    /var/lib/libvirt/images/GuestVM1-vol1
 Vol1            /var/lib/libvirt/images/Vol1
```

For more information about available commands for IBM Z, see *Linux on Z and LinuxONE: KVM Virtual Server Management*, [SC34-2752](#).

6.1.2 Virtual Machine Manager

Virtual Machine Manager (virt-manager) is a graphical user interface (GUI) that you can use to manage VMs through libvirt. It covers the following areas:

- ▶ Monitoring

Virt-manager shows an overview of performance and use statistics for each VM and their CPU, memory, and I/O.

- ▶ Life-cycle management

You can use virt-manager to create, start, stop, change, and delete a VM. It provides graphical wizards for creating a VM.

- ▶ Resource management

You can use virt-manager to define, modify, or delete virtual hardware resources for VMs.

- ▶ Control

Provides easy access to a console of a running VM from virt-manager.

Note: The virt-manager GUI can be installed for the following distributions:

- ▶ RHEL: `yum install virt-manager`
- ▶ Ubuntu: `apt-get install virt-manager`
- ▶ SUSE: `zypper install virt-manager`

As shown in Figure 6-1, the virt-manager GUI can connect to a remote KVM through an IP or host name. This feature allows virt-manager to manage several KVM hosts from a single desktop.

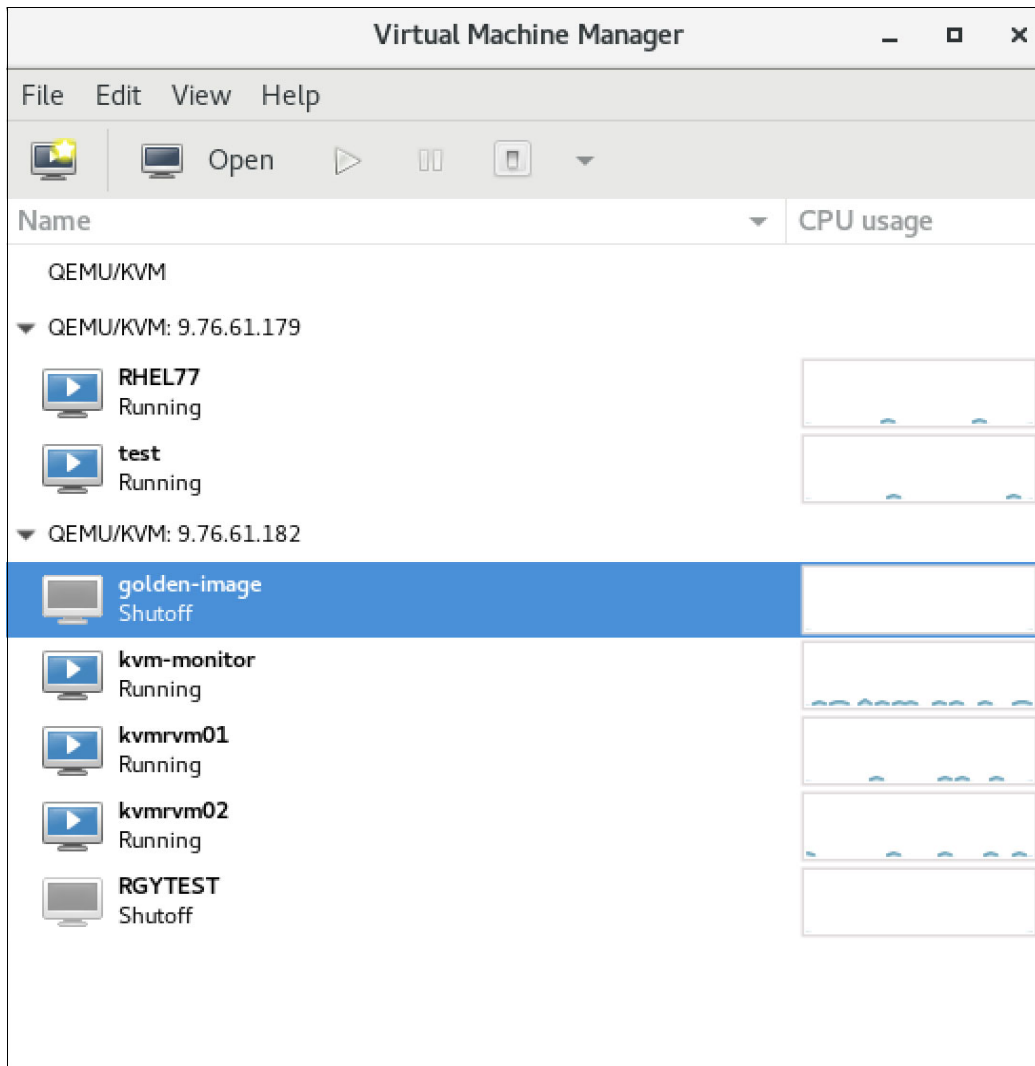


Figure 6-1 Virt-manager with remote KVM host

By using virt-manager, you can define, create, and modify resources that the VM needs, such as disks, network interfaces, and other devices. Some basic resources that virt-manager shows are shown in Figure 6-2.

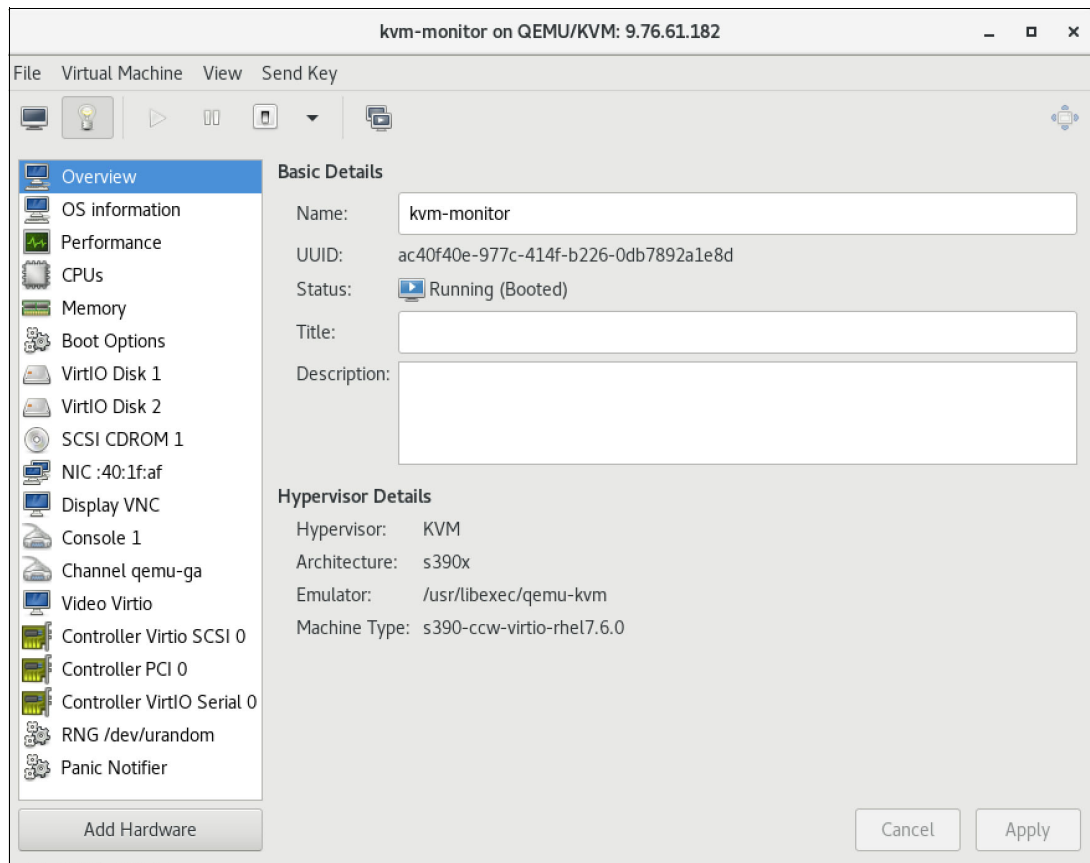


Figure 6-2 virt-manager resources

For more information about virt-manager, see [this website](#).

6.1.3 Cockpit

Cockpit is a web-based interface that is used to easily administrate the servers. Although it does not cover all of the features of the virsh command line, it is an alternative to manage the environment by Web interface instead of acquiring commercial software.

Cockpit concepts extend the features of servers administration by creating plug-ins that can be customized according to business purposes.

Cockpit includes an extension that is called cockpit-machines with which Cockpit can manage, create, and monitor VMs from the web interface that provides the functions:

- ▶ Manage multiple servers in a single Cockpit console
- ▶ Access terminal shell
- ▶ Manage systems resources (memory and CPU)
- ▶ Create, delete, clone VMs
- ▶ Manage system services
- ▶ Add and edit storage

- ▶ Add and edit network interfaces
- ▶ Collect system performance information
- ▶ Install extensions to manage different features, including KVM, Docker, and Kubernetes

Note: Cockpit is available for Red Hat and Ubuntu. It can be installed by using the following commands:

- ▶ RHEL: `yum install cockpit cockpit-machines`
- ▶ Ubuntu: `apt-get install cockpit cockpit-machines`

Installing and using Cockpit

Use the following command to enable Cockpit:

```
sudo systemctl enable --now cockpit.socket
```

Use the following computer or server IP on port 9090 to use Cockpit:

```
https://yourhostname:9090
```

Figure 6-3 shows an example of how to create a VM from the Cockpit console. By using the console, you can connect to the VM shell, list disks, and manage networks and the status of the VM.

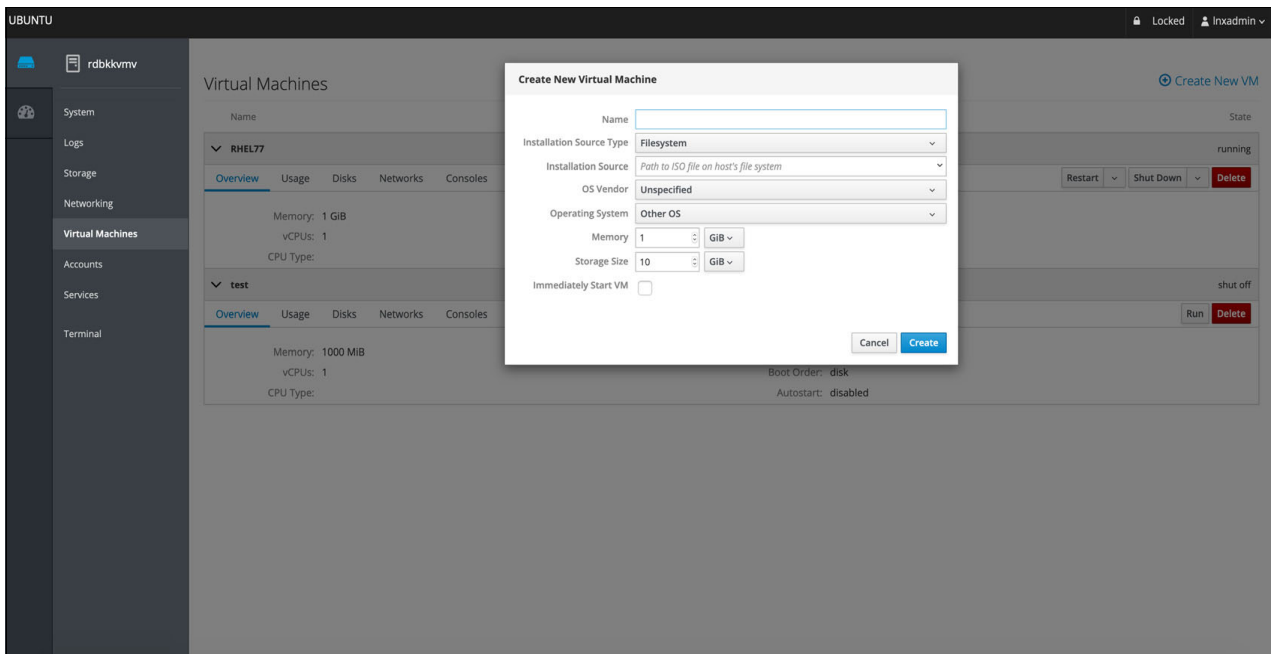


Figure 6-3 Creating a VM

Figure 6-4 shows how to clone a VM by using Cockpit.

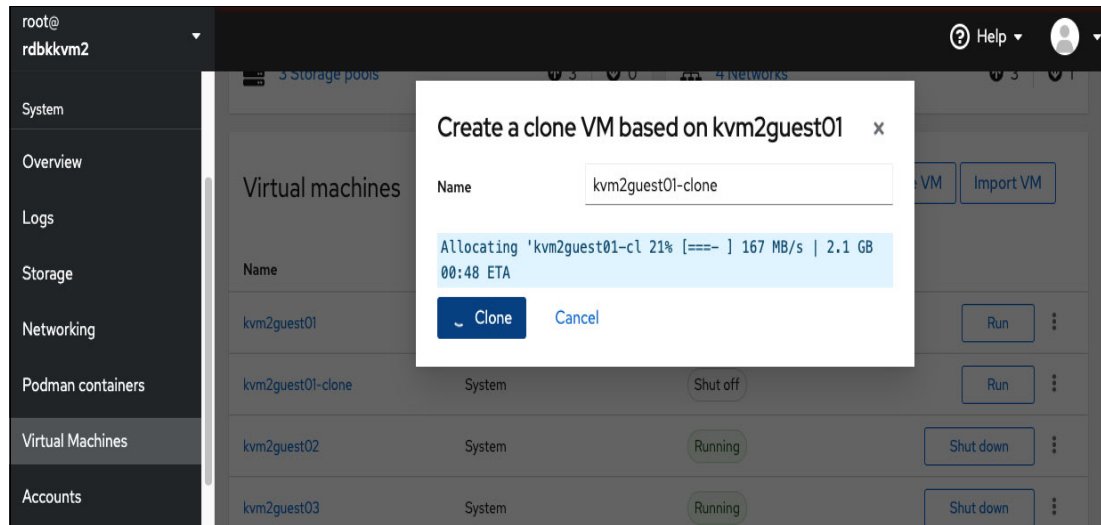


Figure 6-4 Cloning a VM by using Cockpit

The Figure 6-5 shows a sample of creating and attaching a virtual storage volume (GuestVM1-datavol) by using the Cockpit manager console.

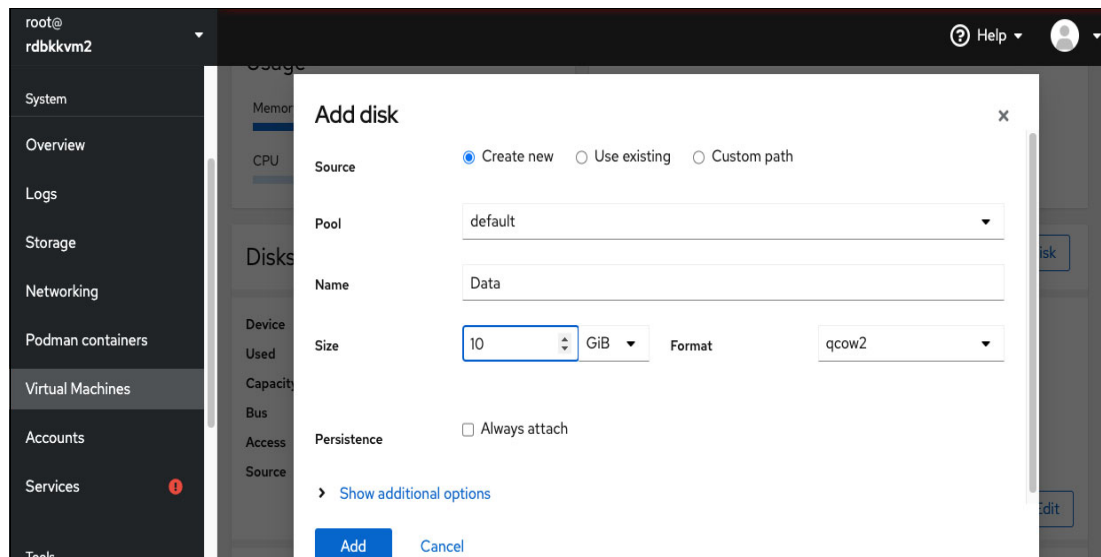


Figure 6-5 Creating a virtual volume for the KVM guest

For more information about Cockpit, see [this website](#).

6.1.4 OpenStack

The OpenStack project is a global collaboration community of developers and cloud computing technologists who are working to create an open source, cloud computing platform for public, private, and hybrid clouds. The cloud computing platform is integrated by a list of interrelated services that provides different management features for the cloud infrastructure.

OpenStack services provide an IaaS or PaaS solution. Each OpenStack service offers an API that facilitates its integration. Based on its service needs, you can install some or all services.

OpenStack can be deployed on IBM Z under the Ubuntu distribution. For the example in this IBM Redbooks publication, OpenStack was installed on Ubuntu 18.04 with an architecture of one controller node and one compute node.

For more information about the full installation process, see [this web page](#).

Note: During the installation process, an error occurred in which the etcd service did not start. To solve this problem, export the variable ETCD_UNSUPPORTED_ARCH by using the following command:

```
export ETCD_UNSUPPORTED_ARCH=s390x
```

The deployed version is Stein that includes the following services:

- ▶ Keystone: Provides authentication and authorizations for all OpenStack services
- ▶ Glance: Provides a catalog and repository for virtual disk images
- ▶ Placement: Provides an API to track resource provider inventories and usages
- ▶ Nova: Provides VMs on demand
- ▶ Neutron: Provides network management
- ▶ Horizon: Provides a web-based user interface

Example 6-4 shows how to create a VM on OpenStack by using the CLI.

Example 6-4 Creating a server on OpenStack

```
root@rdbkvm4:~# openstack server create --flavor m1.small10 --image bionicCloud --nic
net-id=provider --security-group test --key-name mykey provider-instance2Z
```

Field	Value
OS-DCF:diskConfig	MANUAL
OS-EXT-AZ:availability_zone	
OS-EXT-SRV-ATTR:host	None
OS-EXT-SRV-ATTR:hypervisor_hostname	None
OS-EXT-SRV-ATTR:instance_name	
OS-EXT-STS:power_state	NOSTATE
OS-EXT-STS:task_state	scheduling
OS-EXT-STS:vm_state	building
OS-SRV-USG:launched_at	None
OS-SRV-USG:terminated_at	None
accessIPv4	
accessIPv6	
addresses	
adminPass	eKQuE5sb8FMD
config_drive	
created	2019-12-02T15:12:35Z
flavor	m1.small10 (2)
hostId	
id	d2ae87dc-5a27-49f9-a894-6230d85f4470
image	bionicCloud (d7cc628b-3e1a-473c-afd6-e25518f20b60)
key_name	mykey name provider-instance2Z
progress	0
project_id	530e2bfeafd4d8d86de1bd914ed6a36
properties	
security_groups	name='9319eea2-0ae5-4d84-bf27-af57c7a447db'
status	BUILD
updated	2019-12-02T15:12:35Z

```
| user_id | a5c3201248b444e3adc6f83e340a9f60 |
| volumes_attached | |
+-----+-----+
```

A VM also can be created by using the dashboard, as shown in Figure 6-7.

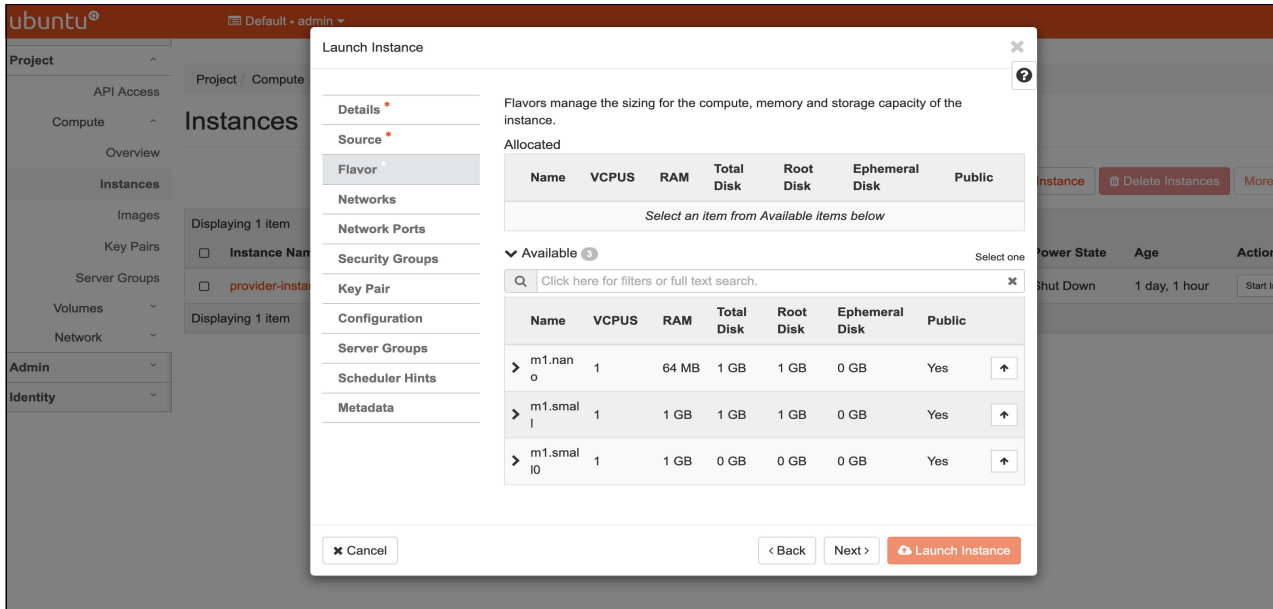


Figure 6-6 Creating a VM in OpenStack through Dashboard

6.1.5 Choosing the correct tool

Each of the tools that are described in this publication (see Table 6-1 on page 197) includes its own advantages. Each advantage features different considerations, depending on the needs and the distribution that is used:

- ▶ Virsh is a shell around libvirt that can be powerful, but it is only a CLI tool and can be complex for some tasks. It is supported on all the distributions that are presented in this IBM Redbooks publication.
- ▶ VM Manager (VMM) provides many functions through a GUI. It can be used to connect to multiple KVM hosts. It is supported on all the distributions that are presented in this publication.
- ▶ Cockpit is a powerful and intuitive tool to work with servers; however, on the KVM side, it is not as complete as Virsh or VMM. Cockpit-machines are more oriented to creating and monitoring a VM, but other complex tasks are limited. Cockpit is supported on RHEL and Ubuntu.
- ▶ OpenStack is a complete and open solution to manage VMs and all of the different resources that are needed around it, such as networking, storage, and security. As of this writing, OpenStack is supported on Ubuntu.

Table 6-1 Resource management tools overview

Task	Virsh	VMM	Cockpit	OpenStack
Interface	CLI	GUI	GUI	GUI, CLI
Access method	SSH client or console	Linux X Window System application	Web browser	Web browser, SSH client, or API
Manage VM lifecycle	Yes	Yes	Yes	Yes
Configure resources available to KVM	Yes	Yes	No	No
Add ECKD volumes to KVM	Yes	No	No	No
Add resources to VMs	Yes	Yes	No	Yes
Overview system performance	No	Yes	Yes	Yes
Console access	VM Console	VM Console	VM Console	VM Console
Distribution	RHEL, Ubuntu, and SUSE	RHEL, Ubuntu, and SUSE	RHEL and Ubuntu	Ubuntu

6.2 Recovery management

After an environment is set up, it is important to protect the data that it contains. Therefore, backup and restore procedures must be part of installing any KVM and Linux environment.

Approaches for backup depend on many varying requirements, including the following examples:

- ▶ What to back up

Do you back up data only or also operating system and middleware or database configuration files? Is it easier to reinstall and modify only several configuration files or is an image copy of the whole disk easier?

- ▶ Recovery objective

What type of risk is acceptable? Several methods are available to recover from a crash state by using journals or must a full backup be available at any time?

- ▶ Recovery time

Is it acceptable if a recovery takes days (applying journals on large databases)? If you are recovering an operating system, is it faster to reinstall or recover?

- ▶ On which level the backup is occurring

In a virtualized environment, disk images of VMs can be backed up from the hypervisor. However, if the VM is active, data in its cache is not backed up. Is it acceptable to shut down the VM for backup or use tools inside it to perform a backup?

- ▶ Different tools for different purposes

Tools are available to back up disk images, and other tools to create file-level backups. Database systems have their own backup solutions because they understand what is occurring inside the database system.

6.2.1 Snapshot

A snapshot is a copy of the VM disk at a specific time. It is useful to take a snapshot of a VM before changing to the VM because a VM can be restored to the same state it was when the snapshot was taken. However, a snapshot alone does not provide a backup.

The Example 6-5 shows the process of creating a snapshot by using `virsh` commands. Remember that `qcow2` storage must be used for snapshots to work. If a snapshot is taken while the VM is running, the snapshot takes only the state of the disk, *not* the state of the memory.

Example 6-5 Creating a snapshot with virsh

```
root@rdbkvm2:/home/lnxadmin# virsh snapshot-create-as --domain RHEL84 --name  
"snapshotRedbook" --description "Snapshot before upgrading"  
Domain snapshot snapshotRedbook created
```

```
root@rdbkvm2:/home/lnxadmin# virsh snapshot-list --domain RHEL84  
Name                Creation Time          State  
-----  
snapshotRedbook    2019-11-27 11:20:58 -0500  running
```

Example 6-6 shows how to revert a VM to a specific snapshot. As shown in the example, the VM is running again after the snapshot is restored.

Example 6-6 Reverting to a snapshot with virsh

```
root@rdbkvm2:/home/lnxadmin# virsh shutdown --domain RHEL84  
Domain RHEL77 is being shutdown
```

```
root@rdbkvm2:/home/lnxadmin# virsh snapshot-revert --domain RHEL84  
--snapshotname snapshotRedbook --running
```

```
root@rdbkvm2:/home/lnxadmin# virsh list  
Id  Name      State  
-----  
1   RHEL84   running
```

6.2.2 Compressing data and backup

One of the frequent administration tasks for Linux and the application is to compress the logs to save disk space and backup them daily by using a backup tool. Several backup tools for Linux are available in commercial and open software.

The IBM z15 machine features a new hardware feature that is called Integrated Accelerator for zEnterprise Data Compression (zEDC) to accelerate file compression. zEDC enables Linux on IBM Z and LinuxONE machines to run the `gzip` compression in the IFL processors and offload the main processors.

To enable the supported Linux versions (RHEL 8.2, SLE15, Ubuntu 20.04, and newer), the only step is to set an environment variable that controls which compression levels must be accelerated. To permanently enable the feature, add this information in the `/etc/bashrc` directory.

The environment variable `DFLTCC_LEVEL_MASK` includes a binary value. Each bit corresponds to one compression level, including the following examples:

- ▶ `DFLTCC_LEVEL_MASK=0x0000` (disable hardware acceleration)
- ▶ `DFLTCC_LEVEL_MASK=0x0002` (enable hardware acceleration for compression level 1)
- ▶ `DFLTCC_LEVEL_MASK=0x007e` (enable hardware acceleration for compression level 1 - 6)
- ▶ `DFLTCC_LEVEL_MASK=0x01ff` (enable hardware acceleration for compression level 0 - 8)

Starting with the last two lines of Example 6-7, the `gzip` compression and zero main processor consumption are greatly improved.

Example 6-7 LAB Test1: Setting up level without acceleration

```
[root@rdbkvm2 isos]# time gzip operlog1.txt
real 1m23.633s
user 1m21.878s
sys 0m1.673s
[root@rdbkvm2 isos]# ls -alh operlog1.txt.gz
-rw-r--r--. 1 root root 467M Nov 22 08:23 operlog1.txt.gz
```

The system spent 1 minute and 23 seconds to complete the compression and allocated 100% of CPU(IFL) for 1 minute and 23 seconds (see Example 6-8).

Example 6-8 Output of top command

```
[root@rdbkvm2 /]# top
PID USER      PR  NI  VIRT  RES  SHR S  %CPU  %MEM    TIME+  COMMAND
407160 root        20   0   3064   1716  1148 R  100.0   0.0   1:23.12  gzip
```

Example 6-9 shows the improvements in processing time when hardware acceleration is used.

Example 6-9 LAB test2: Setting up level by using hardware acceleration

```
[root@rdbkvm2 isos]# time gzip operlog1.txt
Real 0m1.883s
user 0m0.430s
sys 0m1.453s
[root@rdbkvm2 isos]# ls -alh operlog1.txt.gz
-rw-r--r--. 1 root root 621M Nov 22 08:23 operlog1.txt.gz
```

The system spent 1.8 seconds to complete the compression of the same file and did not allocate CPU(IFL) time. However, the compression tax is a small (15%).

6.2.3 IBM FlashCopy

IBM FlashCopy is supported by many IBM Storage subsystems. With the FlashCopy function, the data on target volumes is replaced by data from source volumes when the copy operation starts. FlashCopy can be referred to by other names, including *Time-Zero copy* (T 0), *point-in-time copy*, or *snapshot copy*.

The primary objective of FlashCopy is to create a copy of a source volume on the target volume. This copy is called a *point-in-time copy*. Access to the point-in-time copy of the data on the source volume is through reading the data from the target volume. The point-in-time data that is read from the target volume might not be physically stored on the target volume.

When a FlashCopy relationship is established (more specifically, when the initialization process for a FlashCopy is established and started by using the **FCESTABL** command), the point-in-time data is available for reading from the target volume.

However, if data is written to a track that is a target track in a FlashCopy relationship and the updated target track is read later, the data that is returned is user-updated data and *not* the point-in-time source track data. Target tracks are withdrawn from a FlashCopy relationship when an application writes to these tracks.

The FlashCopy is a feature of the DS8000 storage family and can be used when backing up a Linux guest that is running under KVM on its own SCSI LUN. You use the FlashCopy feature on an IBM SAN Volume Controller to create an exact copy of the device on which the guest is running by using FlashCopy.

Example 6-10 shows how to connect to a SAN Volume Controller and create a FlashCopy. For this example, a FlashCopy of the disk 0110 is created into the disk 0111.

Example 6-10 Creating a FlashCopy

```
[root@localhost dscli]# dscli -hmc1 IP-user USER -passwd PASSWORD

dscli> lsfbvol
Date/Time: December 5, 2019 6:17:05 AM EST IBM DSCLI Version: 7.8.50.497 DS:
IBM.2107-75KCG71
Name          ID   accstate  datastate  configstate  deviceMTM  datatype  extpool  cap
(2^30B)  cap (10^9B)  cap (blocks)
=====
=====
RB_KVM_ARIES_ 0005 Online   Normal    Normal      2107-900  FB 512   P2
40.0          -      83886080
RB_KVM_ARIES_ 0006 Online   Normal    Normal      2107-900  FB 512   P2
40.0          -      83886080
RB_KVM_ARIES_ 0007 Online   Normal    Normal      2107-900  FB 512   P2
40.0          -      83886080
RB_KVM_ARIES_ 0008 Online   Normal    Normal      2107-900  FB 512   P2
40.0          -      83886080
RB_KVM_ARIES_ 0009 Online   Normal    Normal      2107-900  FB 512   P2
40.0          -      83886080
RB_KVM_ARIES_ 000A Online   Normal    Normal      2107-900  FB 512   P2
40.0          -      83886080
RB_KVM_ARIES_ 000B Online   Normal    Normal      2107-900  FB 512   P2
40.0          -      83886080
RB_KVM_ARIES_ 000C Online   Normal    Normal      2107-900  FB 512   P2
40.0          -      83886080
RB_KVM_ARIES_ 000D Online   Normal    Normal      2107-900  FB 512   P2
40.0          -      83886080
RB_KVM_ARIES_ 000E Online   Normal    Normal      2107-900  FB 512   P2
40.0          -      83886080
RB_KVM_ARIES_ 0108 Online   Normal    Normal      2107-900  FB 512   P3
40.0          -      83886080
RB_KVM_ARIES_ 0109 Online   Normal    Normal      2107-900  FB 512   P3
40.0          -      83886080
RB_KVM_ARIES_ 010A Online   Normal    Normal      2107-900  FB 512   P3
40.0          -      83886080
RB_KVM_ARIES_ 010B Online   Normal    Normal      2107-900  FB 512   P3
40.0          -      83886080
```



```

RB_KVM_ARIES_ 010C Online Normal Normal 2107-900 FB 512 P3
40.0 - 83886080
RB_KVM_ARIES_ 010D Online Normal Normal 2107-900 FB 512 P3
40.0 - 83886080
RB_KVM_ARIES_ 010E Online Normal Normal 2107-900 FB 512 P3
40.0 - 83886080
RB_KVM_ARIES_ 010F Online Normal Normal 2107-900 FB 512 P3
40.0 - 83886080
RB_KVM_ARIES_ 0110 Online Normal Normal 2107-900 FB 512 P3
40.0 - 83886080
RB_KVM_ARIES_ 0111 Online Normal Normal 2107-900 FB 512 P3
40.0 - 83886080

```

```

dsccli> mkflash -dev IBM.2107-75KCG71 0110:0111
Date/Time: December 5, 2019 6:19:50 AM EST IBM DSCLI Version: 7.8.50.497 DS:
IBM.2107-75KCG71
CMUC00137I mkflash: FlashCopy pair 0110:0111 successfully created.

```

While the FlashCopy is being made, the status of the process can be seen by using the **lsflash -dev IBM.2107-75KCG71 -l 0110:0111** command.

After the FlashCopy is created on the disk, you can restore it by copying back the contents of the copied disk into the source disk. This process can be done by using FlashCopy, but the direction of the copy is swapped.

6.3 Security management

Security is a key aspect of every IT environment. Ignoring good security practices can lead to a breach and make a platform ineligible for consideration for specific workloads.

The tools and facilities that are used to secure a KVM environment are the same tools that you might use for the Linux VMs. The components that are described in this section do not overlap, but address different facets of the overall security needs, such as authentication, auditing, and network access control.

The following tools are described in this section:

- ▶ FreeIPA
- ▶ sVirt
- ▶ App Armor
- ▶ Linux Audit

6.3.1 FreeIPA

FreeIPA is an integrated security Information Management solution that combines multiple web-based and command-line administration tools. It provides a centralized authentication, authorization, and account information by storing data about users, groups hosts, and other objects. It is built on Open Source components and protocols to ease the management and automation of configuration tasks.

FreeIPA features the following components:

- ▶ 389 Directory Server
 - An Open Source LDAP server for Linux. It stores identities, groups, and organization data.

► Kerberos

A computer-network authentication protocol that is based on tickets allows communication between nodes over a non-secure network, which proves their identity to one another in a secure manner.

► Dogtag Certificate System

Dogtag Certificate System is an enterprise-class certificate authority. It supports all aspects of certificate lifecycle management.

► System Security Services Daemon

Provides a set of daemons to manage access to the different FreeIPA components, such as LDAP and Kerberos.

FreeIPA features the following benefits:

- Centralize identities in one place
- Apply policies to multiple machines uniformly and at the same time
- Set different access levels for users and groups
- Reduce risks of passwords being written down or stored insecurely

After installing the FreeIPA packages, the command **ipa-server-install** guides you through the configuration. Example 6-11 shows a basic configuration for the tool.

Example 6-11 FreeIPA server configuration

```
[root@server ~]# ipa-server-install
```

```
The log file for this installation can be found in /var/log/ipaserver-install.log
```

```
=====
```

```
This program will set up the IPA Server.  
Version 4.7.1
```

```
This includes:
```

- * Configure a stand-alone CA (dogtag) for certificate management
- * Configure the NTP client (chronyd)
- * Create and configure an instance of Directory Server
- * Create and configure a Kerberos Key Distribution Center (KDC)
- * Configure Apache (httpd)
- * Configure the KDC to enable PKINIT

```
To accept the default shown in brackets, press the Enter key.
```

```
Do you want to configure integrated DNS (BIND)? [no]:
```

```
Enter the fully qualified domain name of the computer  
on which you're setting up server software. Using the form  
<hostname>.<domainname>  
Example: master.example.com.
```

```
Server host name [server.redbook.com]:
```

```
The domain name has been determined based on the host name.
```

```
Please confirm the domain name [redbook.com]:
```

```
The kerberos protocol requires a Realm name to be defined.
```

This is typically the domain name converted to uppercase.

Please provide a realm name [REDBOOK.COM]:

Certain directory server operations require an administrative user. This user is referred to as the Directory Manager and has full access to the Directory for system management tasks and will be added to the instance of directory server created for IPA. The password must be at least 8 characters long.

Directory Manager password:
Password (confirm):

The IPA server requires an administrative user, named 'admin'. This user is a regular system account used for IPA server administration.

IPA admin password:
Password (confirm):

The IPA Master Server will be configured with:

Hostname: server.redbook.com
IP address(es): 9.76.61.189
Domain name: redbook.com
Realm name: REDBOOK.COM

The CA will be configured with:

Subject DN: CN=Certificate Authority,0=REDBOOK.COM
Subject base: O=REDBOOK.COM
Chaining: self-signed

Continue to configure the system with these values? [no]: **yes**

The following operations may take some minutes to complete. Please wait until the prompt is returned.

Synchronizing time

No SRV records of NTP servers found and no NTP server or pool address was provided.

Using default chrony configuration.

Attempting to sync time with chronyc.

Process chronyc waitsync failed to sync time!

Unable to sync time with chrony server, assuming the time is in sync. Please check that 123 UDP port is opened, and any time server is on network.

Warning: IPA was unable to sync time with chrony!

Time synchronization is required for IPA to work correctly

Configuring directory server (dirsrv). Estimated time: 30 seconds

Done configuring directory server (dirsrv).

Done configuring Kerberos KDC (krb5kdc).

Configuring kadmin

Done configuring kadmin.

Configuring ipa-custodia

Done configuring ipa-custodia.

Configuring certificate server (pki-tomcatd). Estimated time: 3 minutes

Done configuring certificate server (pki-tomcatd).

Configuring directory server (dirsrv)

Done configuring directory server (dirsrv).
Configuring ipa-otpd
Done configuring ipa-otpd.
Configuring the web interface (httpd)
Done configuring the web interface (httpd).
Configuring Kerberos KDC (krb5kdc)
Done configuring Kerberos KDC (krb5kdc).
Applying LDAP updates
Upgrading IPA:. Estimated time: 1 minute 30 seconds
Done.
Restarting the KDC
Configuring client side components
This program will set up IPA client.
Version 4.7.1

Using existing certificate '/etc/ipa/ca.crt'.
Client hostname: server.redbook.com
Realm: REDBOOK.COM
DNS Domain: redbook.com
IPA Server: server.redbook.com
BaseDN: dc=redbook,dc=com

Configured sudoers in /etc/nsswitch.conf
Configured /etc/sss/sss.conf
Adding SSH public key from /etc/ssh/ssh_host_ecdsa_key.pub
Adding SSH public key from /etc/ssh/ssh_host_ed25519_key.pub
Adding SSH public key from /etc/ssh/ssh_host_rsa_key.pub
Could not update DNS SSHFP records.
SSSD enabled
Configured /etc/openldap/ldap.conf
Configured /etc/ssh/ssh_config
Configured /etc/ssh/sshd_config
Configuring redbook.com as NIS domain.
Client configuration complete.
The ipa-client-install command was successful

After the FreeIPA server is configured, the server can be managed centrally from a web-based interface. Figure 6-7 shows an example of user management.

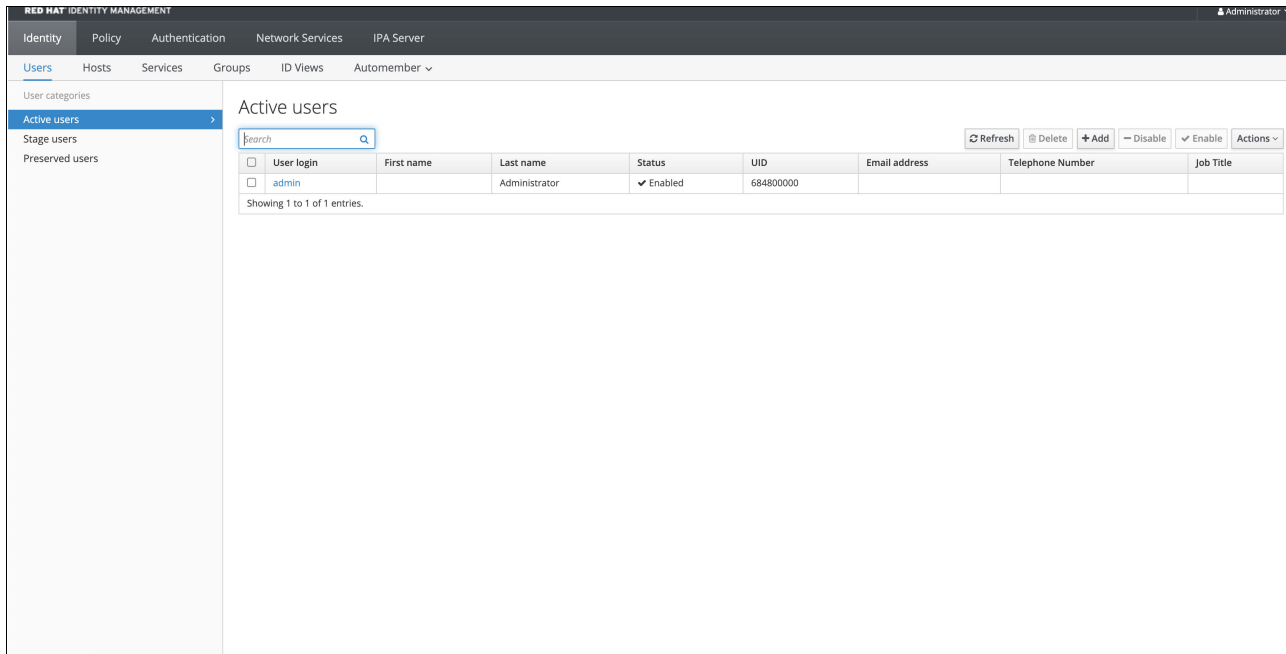


Figure 6-7 FreeIPA console on RHEL

FreeIPA is available for RHEL 8.0 and higher. For more information about FreeIPA, see [this web page](#).

6.3.2 sVirt

The sVirt project is a community effort that is attempting to integrate Mandatory Access Control (MAC) security and Linux-based virtualization (KVM) that is built on SELinux. This integration aims to provide an infrastructure to allow an administrator to define policies for VM isolation.

sVirt ensures that a VM's resources cannot be accessed by any other process (or VM). The system administrator can extend this feature to define fine-grained permissions; for example, to group VMs to share resources.

Example 6-12 shows the virtualization-related Booleans that can be configured.

Example 6-12 sVirt Booleans

```
[root@server ~]# getsebool -a | grep virt
staff_use_svirt --> off
unprivuser_use_svirt --> off
use_virtualbox --> off
virt_read_qemu_ga_data --> off
virt_rw_qemu_ga_data --> off
virt_sandbox_share_apache_content --> off
virt_sandbox_use_all_caps --> on
virt_sandbox_use_audit --> on
virt_sandbox_use_fusefs --> off
virt_sandbox_use_mknod --> off
virt_sandbox_use_netlink --> off
```

```
virt_sandbox_use_sys_admin --> off
virt_transition_userdomain --> off
virt_use_comm --> off
virt_use_execmem --> off
virt_use_fusefs --> off
virt_use_glusterd --> off
virt_use_nfs --> on
virt_use_pcscd --> off
virt_use_rawip --> off
virt_use_samba --> off
virt_use_sanlock --> off
virt_use_usb --> on
virt_use_xserver --> off
```

6.3.3 AppArmor

AppArmor is a Linux kernel security module that is used to confine programs to a limited set of resources. The key to its security model is to bind access control attributes to a program rather than to users. AppArmor is installed and active by default on Ubuntu server.

AppArmor uses profiles of an application to determine what permission it needs. With the package `libvirt-daemon-system`, AppArmor includes profiles that are related to QEMU, KVM, and `libvirtd`. If a VM is created, AppArmor automatically applies the policies to it and enforces those policies.

Example 6-13 shows the profiles that are enforced on an Ubuntu host (`libvirtd` and the VMs that are created are in enforce mode).

Example 6-13 AppArmor status on Ubuntu

```
root@rdbkvm4:/var/lib/libvirt/images# apparmor_status
apparmor module is loaded.
37 profiles are loaded.
35 profiles are in enforce mode.
  /sbin/dhclient
  /usr/bin/evince
  /usr/bin/evince-previewer
  /usr/bin/evince-previewer//sanitized_helper
  /usr/bin/evince-thumbnailer
  /usr/bin/evince//sanitized_helper
  /usr/bin/lxc-start
  /usr/bin/man
  /usr/lib/NetworkManager/nm-dhcp-client.action
  /usr/lib/NetworkManager/nm-dhcp-helper
  /usr/lib/connman/scripts/dhclient-script
  /usr/lib/cups/backend/cups-pdf
  /usr/lib/snapd/snap-confine
  /usr/lib/snapd/snap-confine//mount-namespace-capture-helper
  /usr/sbin/chronyd
  /usr/sbin/cups-browsed
  /usr/sbin/cupsd
  /usr/sbin/cupsd//third_party
  /usr/sbin/ippusbxd
  /usr/sbin/libvirtd
  /usr/sbin/libvirtd//qemu_bridge_helper
```

```

/usr/sbin/tcpdump
libreoffice-senddoc
libreoffice-soffice//gpg
libreoffice-xpdfimport
libvirt-0a01d085-11a7-4cd6-934c-f1f72c0f9a83
libvirt-ca749820-a20a-40f2-9a63-33c72272cc92
libvirt-d2ae87dc-5a27-49f9-a894-6230d85f4470
lxc-container-default
lxc-container-default-cgns
lxc-container-default-with-mounting
lxc-container-default-with-nesting
man_filter
man_groff
virt-aa-helper
2 profiles are in complain mode.
libreoffice-oopslash
libreoffice-soffice
7 processes have profiles defined.
7 processes are in enforce mode.
/usr/sbin/chronyd (2004)
/usr/sbin/cups-browsed (201682)
/usr/sbin/cupsd (201681)
/usr/sbin/libvirtd (2182)
libvirt-0a01d085-11a7-4cd6-934c-f1f72c0f9a83 (60044)
libvirt-ca749820-a20a-40f2-9a63-33c72272cc92 (10999)
libvirt-d2ae87dc-5a27-49f9-a894-6230d85f4470 (145003)
0 processes are in complain mode.
0 processes are unconfined but have a profile defined.

```

6.3.4 Linux Audit

Linux Audit is a security tool that creates audit records for operations that occur within a Linux system. It does *not* protect or prevent problems from occurring on the system; instead, it only logs what occurs. The Linux Audit package is included in all three main distributions; however, it is enabled by default on RHEL and SLES only.

Note: To install `auditd` on Ubuntu, run the `apt-get install auditd` command.

Linux Audit includes the following primary use cases:

- ▶ File access watches
- ▶ Commands that are run by a specific user
- ▶ Recording system call activity
- ▶ Network activity through firewall rich rules
- ▶ Security event recording
- ▶ Searching and reporting on the audit logs

One of the concerns that might prevent the adoption of Linux Audit is the issue of performance. However, the same issue can occur in other operating systems if too high of a level of recording is selected.

Suitable planning and testing can help avoid Linux Audit-related performance issues. Testing also must be done to ensure that the rules are implemented to capture the events that you want recorded and evaluate any potential performance effects.

Rules can be configured on the `/etc/audit/rules.d/` path on all three distributions. When a modification is done, the service must be restarted by using the `systemctl restart auditd` command.

Example 6-14 shows an example of a report of Linux audit.

Example 6-14 Linux audit report

```
rdbkkvms:~ # aureport --summary
```

```
Summary Report
```

```
=====
```

```
Range of time in logs: 20/11/19 17:06:55.350 - 03/12/19 10:08:33.415
```

```
Selected time for report: 20/11/19 17:06:55 - 03/12/19 10:08:33.415
```

```
Number of changes in configuration: 887
```

```
Number of changes to accounts, groups, or roles: 3
```

```
Number of logins: 46
```

```
Number of failed logins: 4
```

```
Number of authentications: 110
```

```
Number of failed authentications: 21
```

```
Number of users: 3
```

```
Number of terminals: 21
```

```
Number of host names: 19
```

```
Number of executables: 10
```

```
Number of commands: 0
```

```
Number of files: 0
```

```
Number of AVC's: 0
```

```
Number of MAC events: 0
```

```
Number of failed syscalls: 0
```

```
Number of anomaly events: 128
```

```
Number of responses to anomaly events: 0
```

```
Number of crypto events: 856
```

```
Number of integrity events: 0
```

```
Number of virt events: 1261
```

```
Number of keys: 0
```

```
Number of process IDs: 449
```

```
Number of events: 4840
```

By using Linux Audit, a report can be built with the virtualization events on the machine. Example 6-15 shows a report of a VM being restarted on the system.

Example 6-15 Linux audit on virtualization

```
root@rdbkvm4:~# aureport --virt
```

Virtualization Report

=====

```
# date time type success event
```

=====

```
1. 12/03/2019 10:03:49 VIRT_CONTROL yes 111
2. 12/03/2019 10:05:31 VIRT_MACHINE_ID yes 119
3. 12/03/2019 10:05:31 VIRT_MACHINE_ID yes 120
4. 12/03/2019 10:05:31 VIRT_RESOURCE yes 124
5. 12/03/2019 10:05:31 VIRT_RESOURCE yes 125
6. 12/03/2019 10:05:32 VIRT_RESOURCE yes 127
7. 12/03/2019 10:05:32 VIRT_RESOURCE yes 128
8. 12/03/2019 10:05:32 VIRT_RESOURCE yes 129
9. 12/03/2019 10:05:32 VIRT_RESOURCE yes 130
10. 12/03/2019 10:05:32 VIRT_RESOURCE yes 131
11. 12/03/2019 10:05:32 VIRT_RESOURCE yes 132
12. 12/03/2019 10:05:32 VIRT_RESOURCE yes 133
13. 12/03/2019 10:05:32 VIRT_RESOURCE yes 134
14. 12/03/2019 10:05:32 VIRT_RESOURCE yes 135
15. 12/03/2019 10:05:32 VIRT_RESOURCE yes 136
16. 12/03/2019 10:05:32 VIRT_RESOURCE yes 137
17. 12/03/2019 10:05:32 VIRT_RESOURCE yes 138
18. 12/03/2019 10:05:32 VIRT_RESOURCE yes 139
19. 12/03/2019 10:05:32 VIRT_RESOURCE yes 140
20. 12/03/2019 10:05:32 VIRT_RESOURCE yes 141
21. 12/03/2019 10:05:32 VIRT_CONTROL yes 142
```



High Availability for IBM General Parallel File System

The IBM General Parallel File System (GPFS) is a cluster file system that provides concurrent access to a single file system or set of file systems from multiple nodes. These nodes can all be SAN attached or a mix of SAN and network attached. This configuration enables high performance access to this common set of data to support a scale-out solution or provides a high availability platform.

This chapter discusses how to set up IBM Spectrum Scale or GPFS for a two-node cluster, Kernel-based Virtual Machine (KVM) Compute hosts. For more information about how to install IBM Spectrum Scale on the Red Hat Operating System, see this [web page](#).

KVM Compute hosts support two types of shared storage: NFS and IBM Spectrum Scale. This IBM Redbooks publication covers the configuration for IBM Spectrum Scale. Throughout this chapter, a step-by-step sequence approach shows the commands and their expected output to achieve the objective of creating the environment.

After the installation is complete, follow the steps to set up IBM Cloud Infrastructure Center that are described in 9.1, “Installing IBM Cloud Infrastructure Center” on page 322.

Note: Consider the following points:

- ▶ Changing a shared storage path after the host is managed is *not* supported.
- ▶ Ensure that the IBM Spectrum Scale or NFS storage server is always active; otherwise, the host fails to be added for timeout exception and the virtual machine (VM) deployment might fail for permission deny.
- ▶ Ensure that the shared storage path always is mounted to the share storage server; otherwise, the local storage of the host is used and the VM deployment might fail for permission deny.

This chapter includes the following topics:

- ▶ “Environment overview” on page 213
- ▶ “Zoning and LUN masking” on page 214
- ▶ “Downloading IBM Spectrum Scale from IBM Fix Central” on page 219
- ▶ “Installing IBM Spectrum Scale” on page 221
- ▶ “Building the GPFS portability layer” on page 225
- ▶ “Handling Linux kernel updates” on page 226
- ▶ “GPFS general configuration” on page 228
- ▶ “Working with the General Parallel File System” on page 241

7.1 Environment overview

Figure 7-1 shows the GPFS folder that is created for our two KVM hosts. The LUN disks are shared with all KVM hosts and the file system is managed by IBM Spectrum Scale.

Note: The LUN that is highlighted in red in Figure 7-1 is used for the tiebreaker disk. How it is configured is described in 7.8.6, “**Setting up a tiebreaker disk**” on page 249.

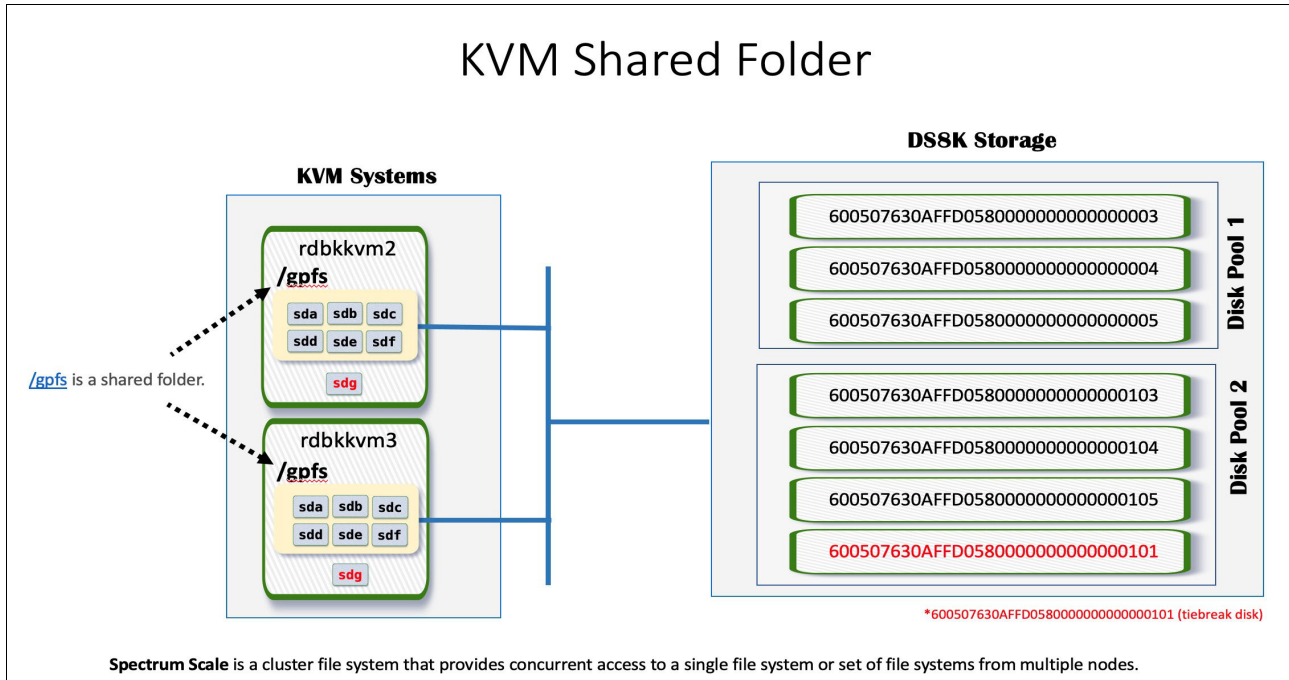


Figure 7-1 KVM Hosts GPFS Shared Folder

7.2 Zoning and LUN masking

In our environment, the SAN infrastructure was defined by using four CHPIDs that are connected to four different FICON directors (switches). As a result, ranges of FCP devices are created by using NPIV, as shown in Figure 7-2.

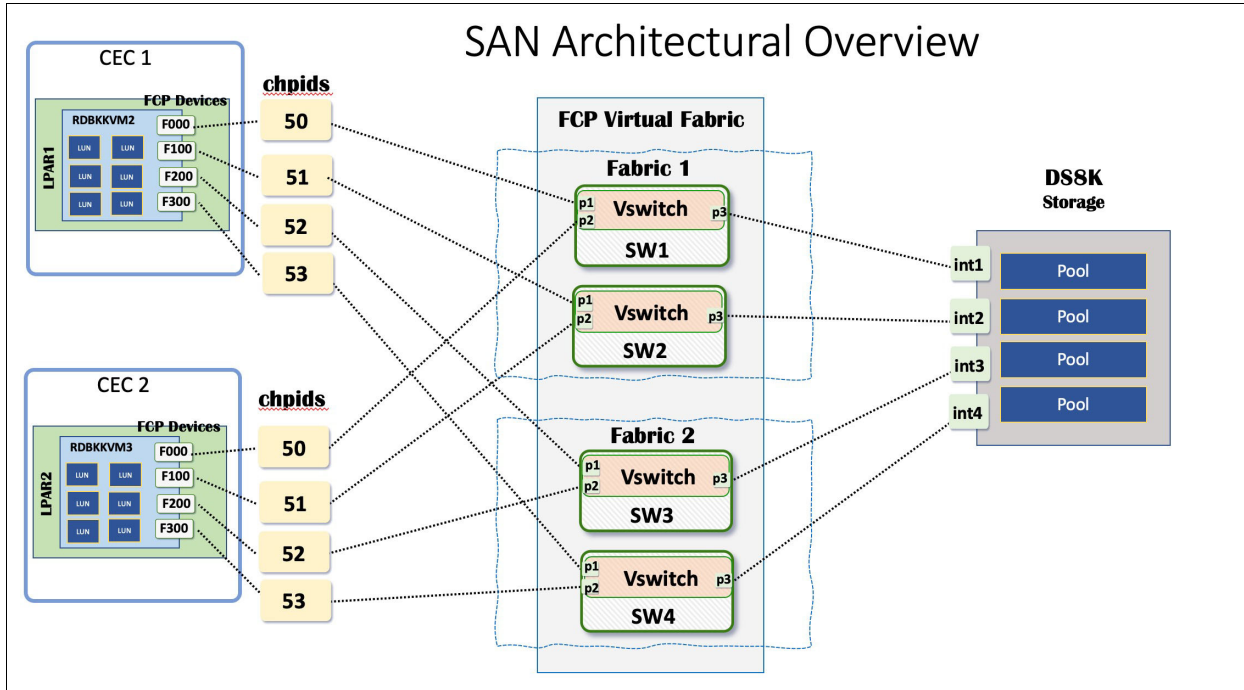


Figure 7-2 SAN infrastructure

The I/O definition of FCP CHPIDs and devices are listed in Table 7-1.

Table 7-1 FCP CHPIDS and devices

z/VM SAN/FCP CHPIDS CEC#1			
CHPIDS	LPARS	ADDRESSES	DEVICES
50	LPAR1	F000-F0EF	240
51	LPAR1	F100-F1EF	240
52	LPAR 1	F200-F2EF	240
53	LPAR1	F300-F3EF	240
z/VM SAN/FCP CHPIDS CEC#2			
CHPIDS	LPARS	ADDRESSES	DEVICES
50	LPAR2	F000-F0EF	240
51	LPAR2	F100-F1EF	240
52	LPAR2	F200-F2EF	240
53	LPAR2	F300-F3EF	240

The following FCP devices were assigned logical WWPNs by using NPIV that was generated by HMC/SE:

- ▶ LPAR1,04,07,50,00,f000,c05076dbdc003000,On,Yes,0168,c05076dbdc001681
- ▶ LPAR1,04,07,51,00,f100,c05076dbdc003300,On,Yes,0105,c05076dbdc001051
- ▶ LPAR2,04,07,50,00,f000,c05076d6a4003000,On,Yes,0168,c05076d6a4001681
- ▶ LPAR2,04,07,51,00,f100,c05076d6a4003300,On,Yes,0105,c05076d6a4001051
- ▶ LPAR2,04,07,52,00,f200,c05076d6a4003600,On,Yes,0170,c05076d6a4001701
- ▶ LPAR2,04,07,53,00,f300,c05076d6a4003900,On,Yes,0229,c05076d6a4002291

Complete the following steps to set up the LUN masking for a DS8K storage:

1. Access the storage (<https://x.x.x.x>, where x.x.x.x represents the Storage IP address) by using your storage administrator credentials.
2. Click the **Hosts** section and then, click **Create Host** (see Figure 7-3).

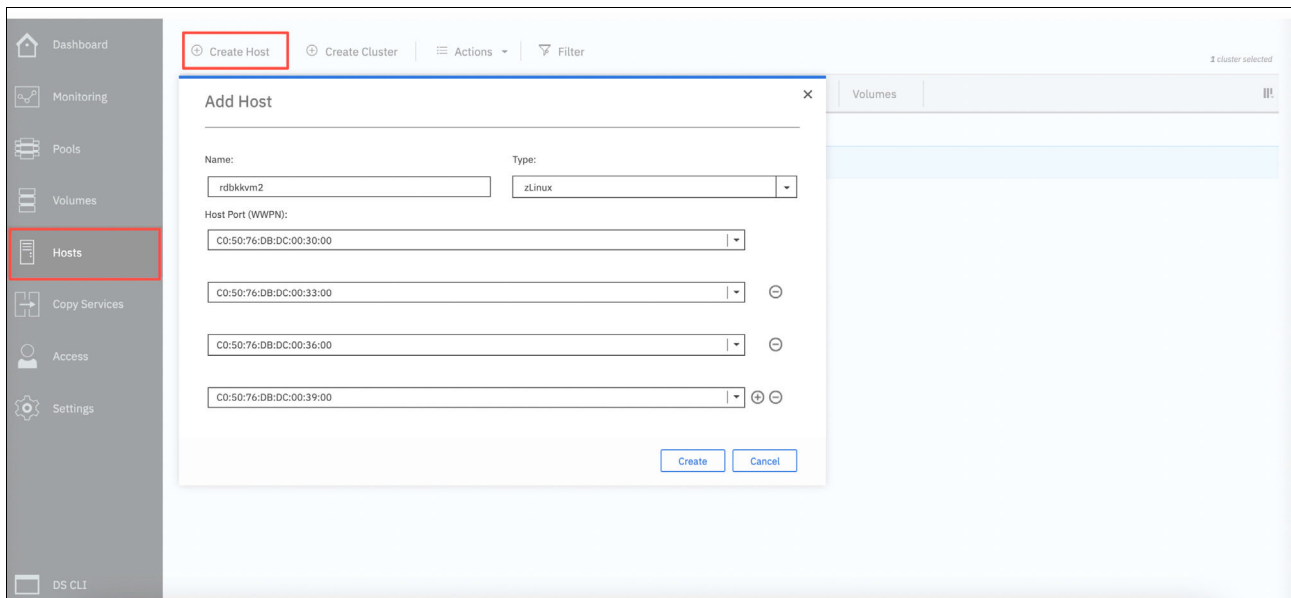


Figure 7-3 Creating a host

3. Complete the form with information about your KVM LPAR name and WWPNs (repeat these steps for all LPARs).

4. Use the filter feature to search for your hosts (see Figure 7-4).

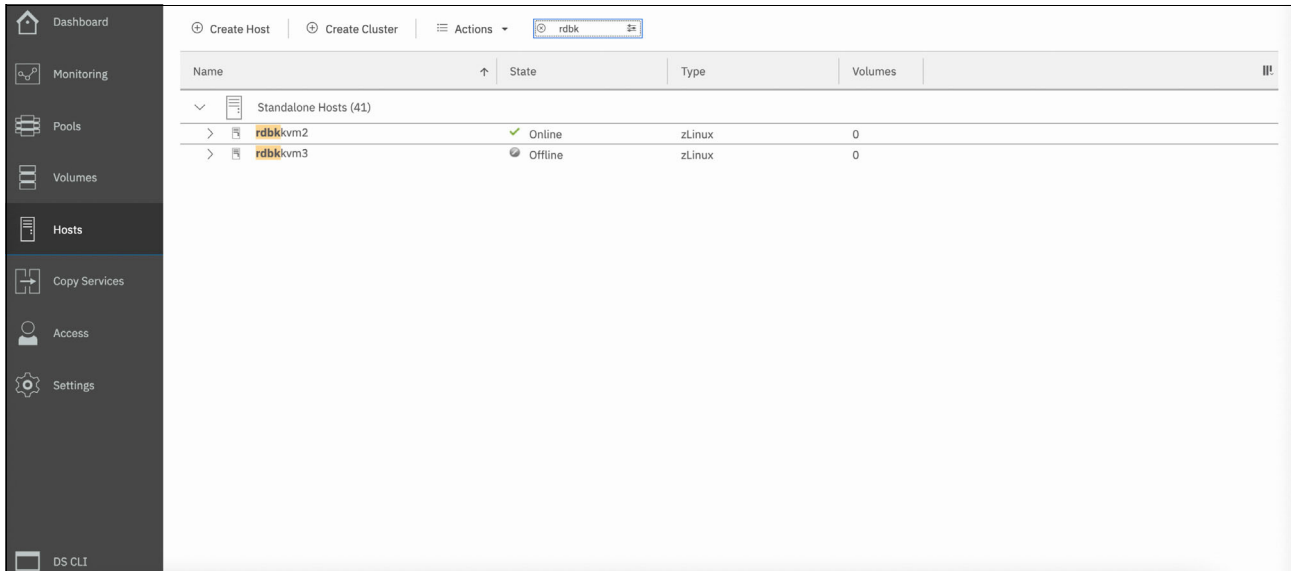


Figure 7-4 Searching your hosts by using filters

5. Click **Create Cluster**. A dialog window opens. Enter the cluster name and then, click **Create** (see Figure 7-5).

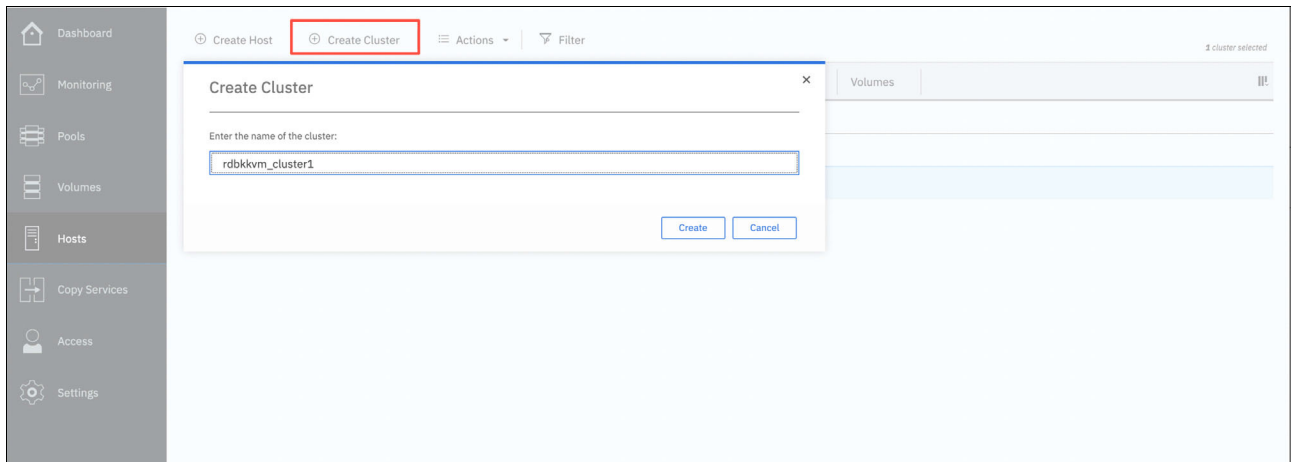


Figure 7-5 Creating a cluster

6. Select the two KVM hosts (see Figure 7-6).

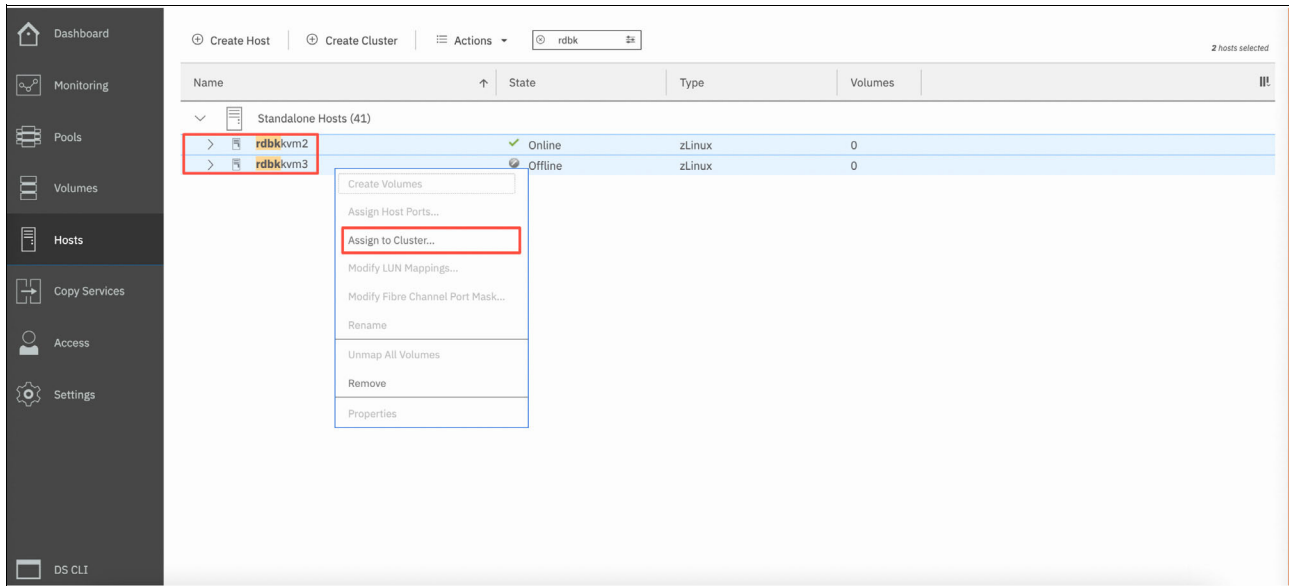


Figure 7-6 Selecting the KVM Host

7. Assign the KVM hosts to the cluster (see Figure 7-7 and Figure 7-8 on page 218).

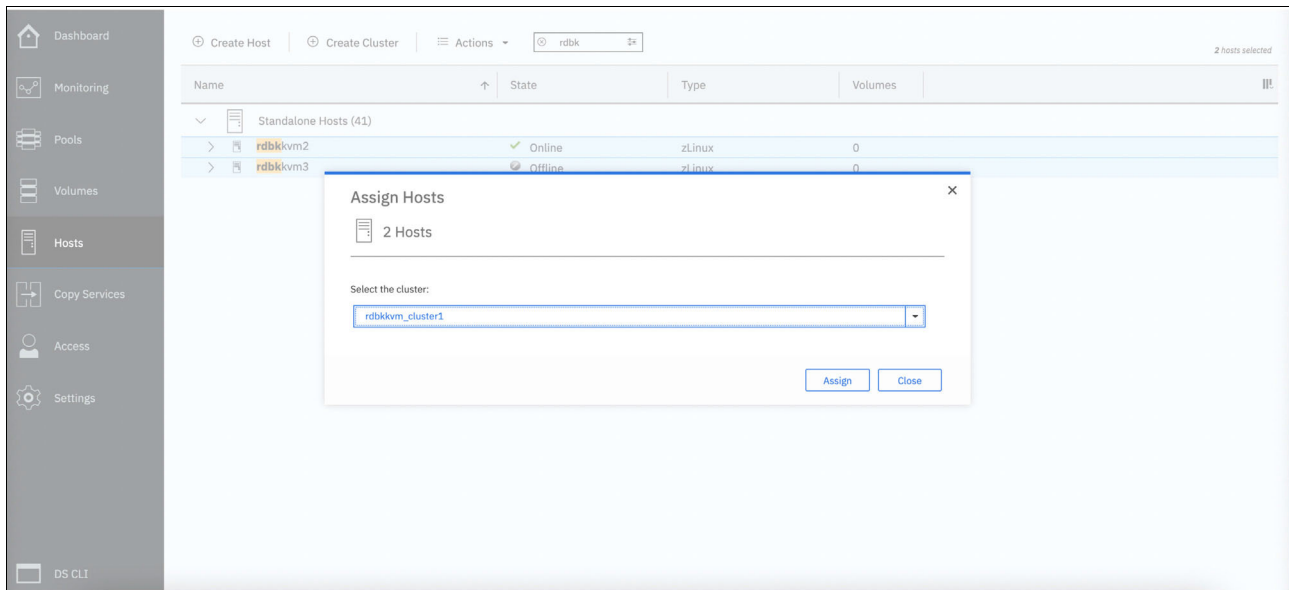


Figure 7-7 Assigning hosts to cluster (1)

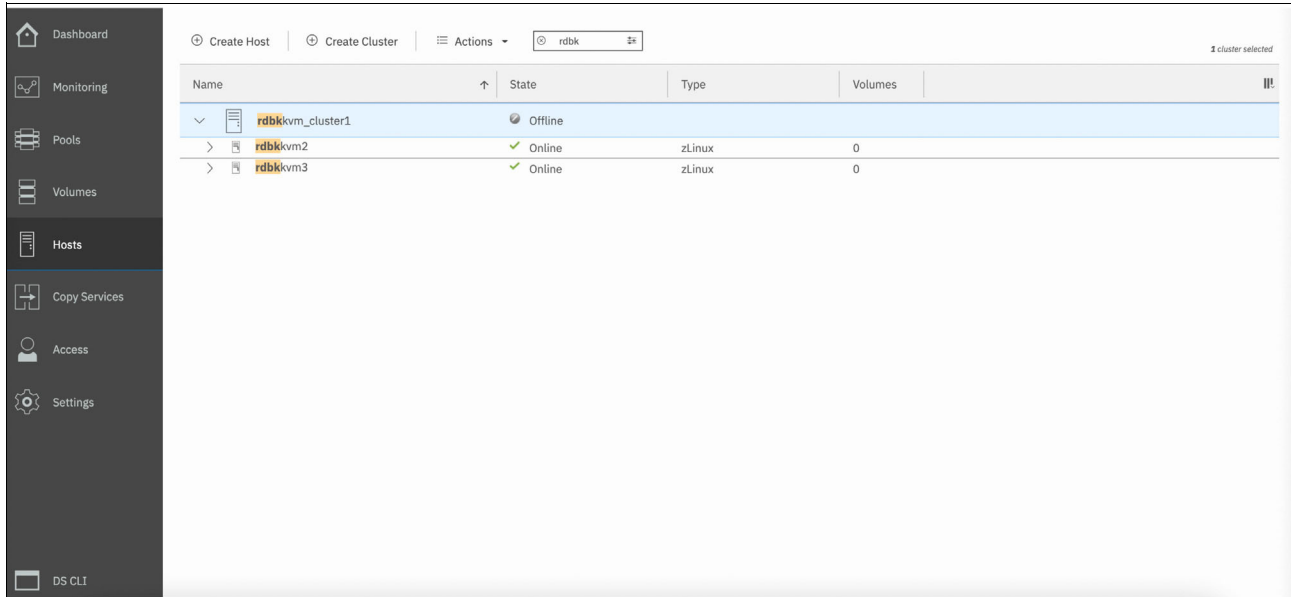


Figure 7-8 Assigning hosts to cluster (2)

- In the Volumes Section on the left side of the window, click **Create Volumes**. Under **Quick Volume Creation**, select **Open Systems**. Then, select the storage pools, enter the Name prefix, enter the wanted disk capacity and then, **Create** (see Figure 7-9).

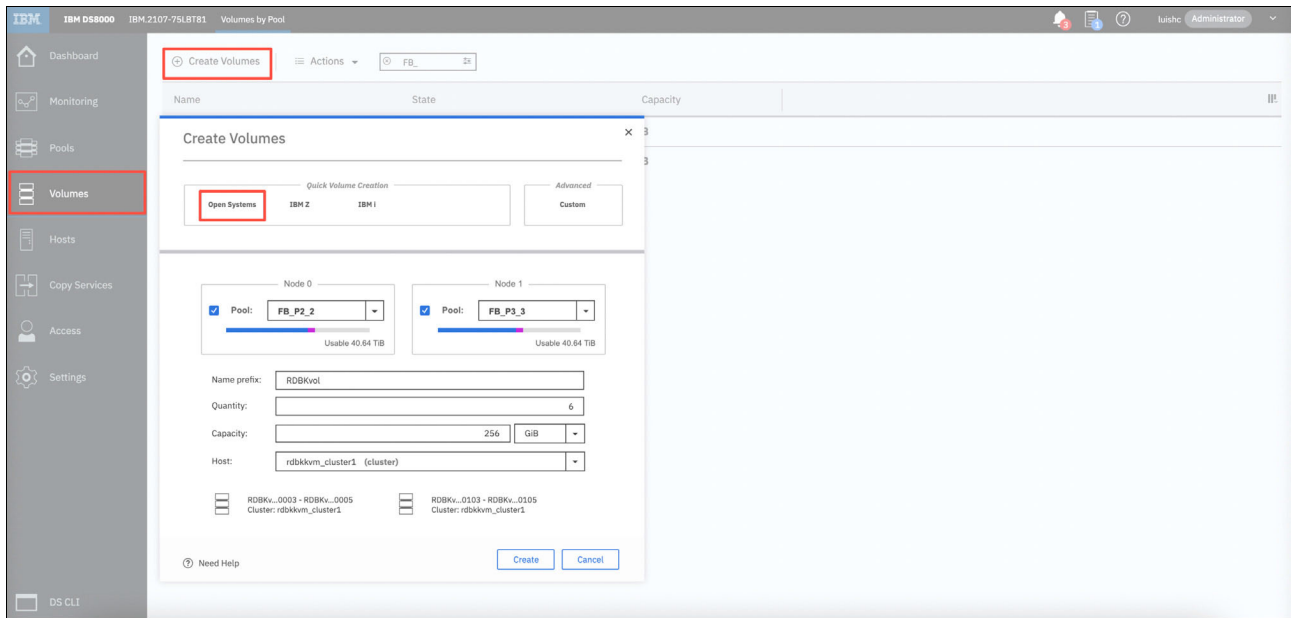


Figure 7-9 Creating volumes and selecting storage pools

9. Confirm that disks were created and assigned to the KVM LPARs, as shown in Figure 7-10.

The screenshot shows a management console interface with a sidebar on the left containing navigation options: Dashboard, Monitoring, Pools, Volumes, Hosts, Copy Services, Access, and Settings. The main area displays a table of disks assigned to KVM LPARs. The table has columns for Name, State, Host Type, LUN, Mapping, Capacity, and Pool. The data is organized into three clusters: rdbkvm_cluster1, rdbkvm2, and rdbkvm3. Each cluster contains several disks with their respective states and configurations.

Name	State	Host Type	LUN	Mapping	Capacity	Pool
rdbkvm_cluster1						
rdbkvm2						
RDBKvol_0003	Normal		40004003	Public	256.0 GiB	FB_P2_2
RDBKvol_0004	Normal		40004004	Public	256.0 GiB	FB_P2_2
RDBKvol_0005	Normal		40004005	Public	256.0 GiB	FB_P2_2
RDBKvol_0103	Normal		40014003	Public	256.0 GiB	FB_P3_3
RDBKvol_0104	Normal		40014004	Public	256.0 GiB	FB_P3_3
RDBKvol_0105	Normal		40014005	Public	256.0 GiB	FB_P3_3
rdbkvm3						
zLinux						
RDBKvol_0003	Normal		40004003	Public	256.0 GiB	FB_P2_2
RDBKvol_0004	Normal		40004004	Public	256.0 GiB	FB_P2_2
RDBKvol_0005	Normal		40004005	Public	256.0 GiB	FB_P2_2
RDBKvol_0103	Normal		40014003	Public	256.0 GiB	FB_P3_3
RDBKvol_0104	Normal		40014004	Public	256.0 GiB	FB_P3_3
RDBKvol_0105	Normal		40014005	Public	256.0 GiB	FB_P3_3

Figure 7-10 Assigning created disks to KVM LPARs

7.3 Downloading IBM Spectrum Scale from IBM Fix Central

Complete the following steps to download IBM Spectrum Scale from IBM Fix Central:

1. Go to <https://www.ibm.com/support/fixcentral/>.
2. Search for the IBM Spectrum Scale product then Select the **Installed Version** and the **Platform** by using the window pull-down menus, then click **Continue**. (see Figure 7-11).

The screenshot shows the IBM Fix Central website interface. At the top, there are navigation links: Support, Downloads, Documentation, Forums, Cases, Monitoring, and Manage support account. Below the navigation, there is a search bar and a 'Find product' button. The main content area contains instructions and a form for product selection. The form includes a 'Product selector' dropdown menu with 'IBM Spectrum Scale' selected, an 'Installed Version' dropdown menu with '5.1.2' selected, and a 'Platform' dropdown menu with 'Linux390 64-bit' selected. A 'Continue' button is located below the form. Red arrows point to the 'Product selector', 'Installed Version', and 'Platform' dropdown menus.

Figure 7-11 IBM Fix Central product selection

- When the **Selected fixes** page appears, scroll down to the **Standard** section then select the latest Spectrum Scale Standard (version) for s390x-Linux. (see Figure 7-12). Then, click **Continue**.

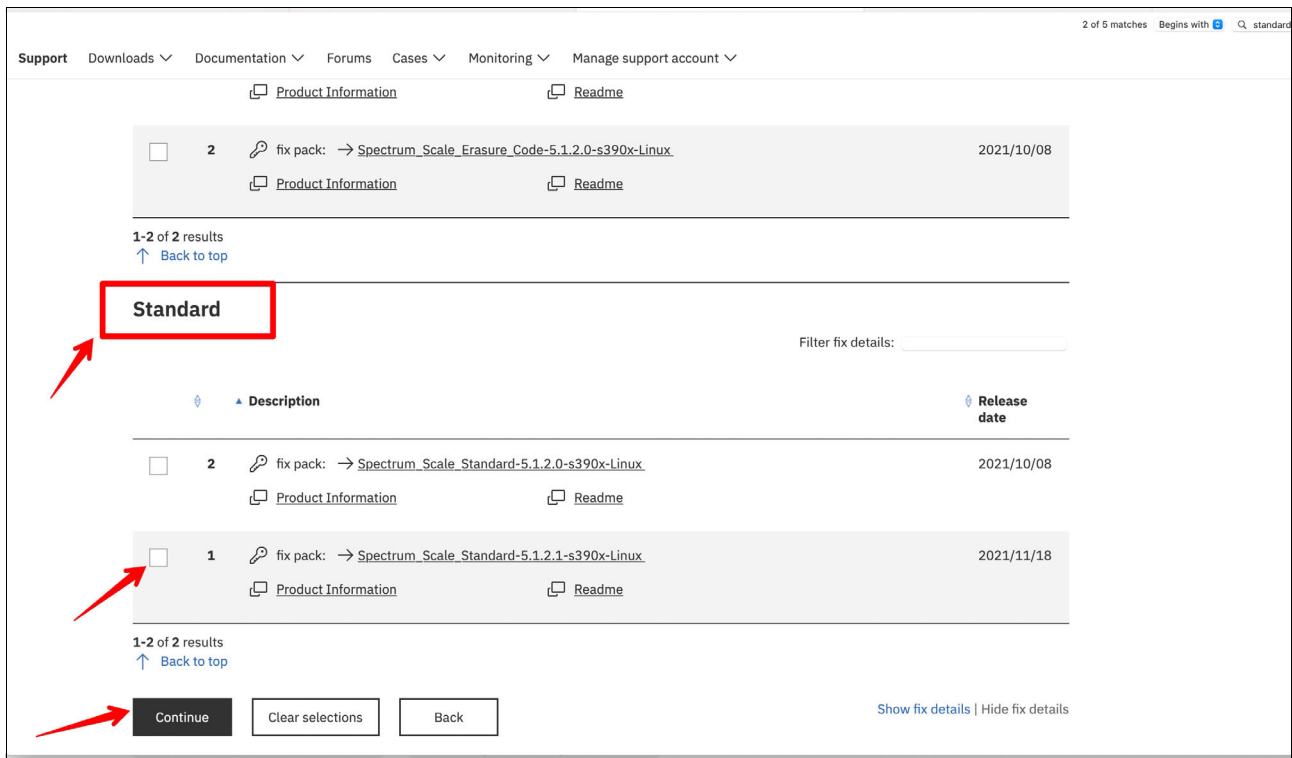


Figure 7-12 Selecting IBM Spectrum Scale

- Select your preferred download method and click **Continue** (see Figure 7-13).

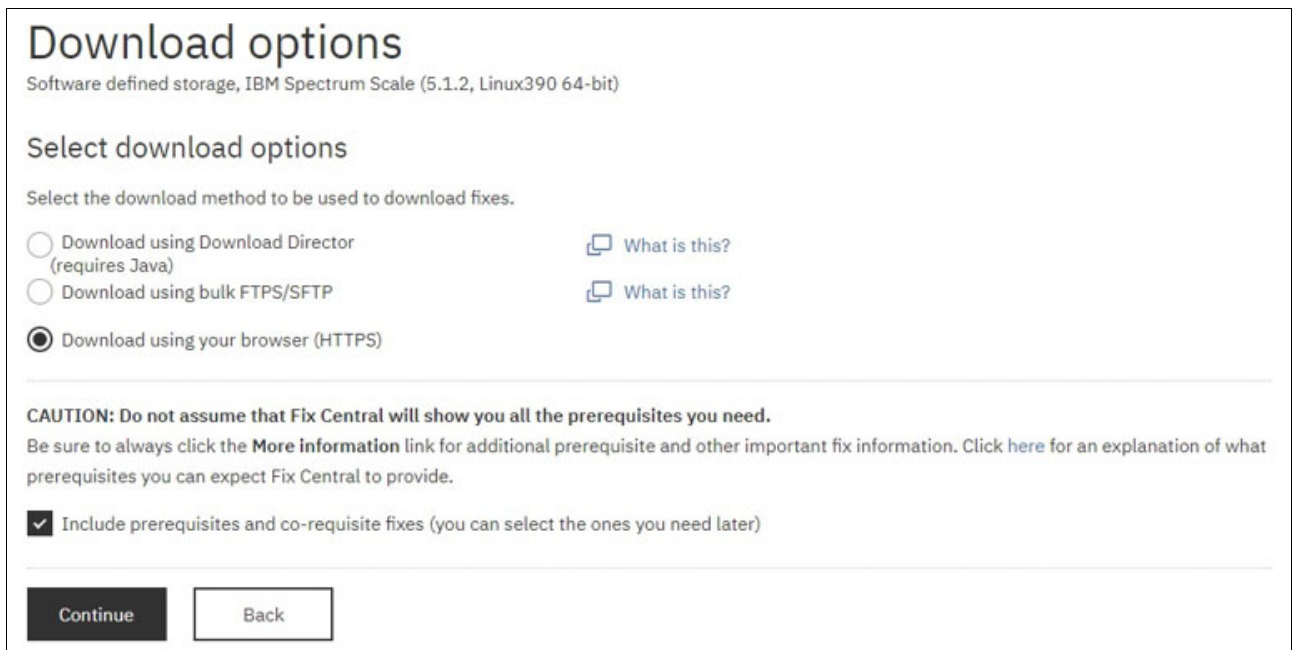


Figure 7-13 Selecting the download method

5. Download all files to a local folder (see Figure 7-14).

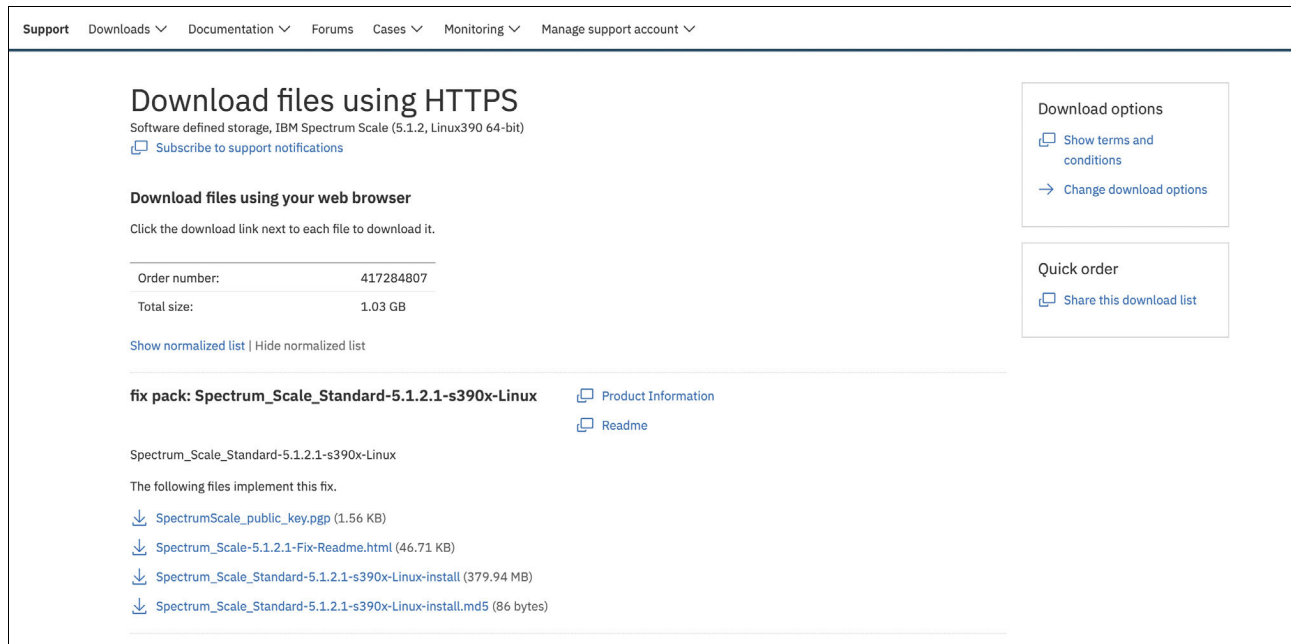


Figure 7-14 Selecting download options

7.4 Installing IBM Spectrum Scale

Complete the following steps to install IBM Spectrum Scale:

1. Create the `/opt/spectrum_gpfs` folder:

```
# mkdir -pv /opt/spectrum_gpfs
```

2. Copy the downloaded file:

```
(Spectrum_Scale_Standard-5.1.2.1-s390x-Linux-install) to /opt/spectrum_gpfs
```

3. Ensure that the following prerequisite Linux packages are installed:

```
# /usr/bin/yum -y install cpp selinux-policy-base selinux-policy-targeted  
nfs-utils boost-regex ethtool rpcbind nfs-utils psmisc iputils m4 ksh  
postgresql-contrib postgresql-server make kernel-devel gcc-c++
```

4. Extract the files and Execute Spectrum Installation on KVM COMPUTE that you are installing (see Example 7-1).

Example 7-1 Installing IBM Spectrum Scale

```
# cd /opt/spectrum_gpfs  
  
# chmod +x Spectrum_Scale_Standard-5.1.2.1-s390x-Linux-install
```

Note: We used `--silent` option below to accept the product license

```
# ./Spectrum_Scale_Standard-5.1.2.1-s390x-Linux-install --silent
```

```
Extracting License Acceptance Process Tool to /usr/lpp/mmfs/5.1.2.1 ...  
tail -n +648 ./Spectrum_Scale_Standard-5.1.2.1-s390x-Linux-install | tar -C  
/usr/lpp/mmfs/5.1.2.1 -xvz --exclude=installer --exclude=*rpms --exclude=*debs  
--exclude=*rpm --exclude=*tgz --exclude=*deb --exclude=*tools* 1> /dev/null
```

Installing JRE ...

If directory /usr/lpp/mmfs/5.1.2.1 has been created or was previously created during another extraction, .rpm, .deb, and repository-related files in it (if there were) will be removed to avoid conflicts with the ones being extracted.

```
tail -n +648 ./Spectrum_Scale_Standard-5.1.2.1-s390x-Linux-install | tar -C
/usr/lpp/mmfs/5.1.2.1 --wildcards -xvz ibm-java*tgz 1> /dev/null
tar -C /usr/lpp/mmfs/5.1.2.1/ -xzf /usr/lpp/mmfs/5.1.2.1/ibm-java*tgz
```

Invoking License Acceptance Process Tool ...

```
/usr/lpp/mmfs/5.1.2.1/ibm-java-s390x-71/jre/bin/java -cp
/usr/lpp/mmfs/5.1.2.1/LAP_HOME/LAPApp.jar com.ibm.lex.lapapp.LAP -l
/usr/lpp/mmfs/5.1.2.1/LA_HOME -m /usr/lpp/mmfs/5.1.2.1 -s /usr/lpp/mmfs/5.1.2.1 -t 5
```

License Agreement Terms accepted.

Extracting Product RPMs to /usr/lpp/mmfs/5.1.2.1 ...

```
tail -n +648 ./Spectrum_Scale_Standard-5.1.2.1-s390x-Linux-install | tar -C
/usr/lpp/mmfs/5.1.2.1 --wildcards -xvz Public_Keys ansible-toolkit ganesha_rpms/rhel7
ganesha_rpms/rhel8 ganesha_rpms/sles15 gpfs_rpms/rhel7 gpfs_rpms/rhel8 gpfs_rpms/sles15
smb_rpms/rhel7 smb_rpms/rhel8 smb_rpms/sles15 tools/repo zimon_rpms/rhel7 zimon_rpms/rhel8
zimon_rpms/sles15 gpfs_rpms manifest 1> /dev/null
```

- Public_Keys
- ansible-toolkit
- ganesha_rpms/rhel7
- ganesha_rpms/rhel8
- ganesha_rpms/sles15
- gpfs_rpms/rhel7
- gpfs_rpms/rhel8
- gpfs_rpms/sles15
- smb_rpms/rhel7
- smb_rpms/rhel8
- smb_rpms/sles15
- tools/repo
- zimon_rpms/rhel7
- zimon_rpms/rhel8
- zimon_rpms/sles15
- gpfs_rpms
- manifest

Removing License Acceptance Process Tool from /usr/lpp/mmfs/5.1.2.1 ...

```
rm -rf /usr/lpp/mmfs/5.1.2.1/LAP_HOME /usr/lpp/mmfs/5.1.2.1/LA_HOME
```

Removing JRE from /usr/lpp/mmfs/5.1.2.1 ...

```
rm -rf /usr/lpp/mmfs/5.1.2.1/ibm-java*tgz
```

```
=====
Product packages successfully extracted to /usr/lpp/mmfs/5.1.2.1
```

7.4.1 Working with clusters and deploying protocols

For more information about the steps that are described in this section, see *IBM Spectrum Scale Concepts, Planning and Installation Guide*, SC28-3161.

The following tasks are described in this section:

- ▶ Install a cluster
- ▶ Upgrade a cluster
- ▶ Add nodes to a cluster
- ▶ Deploy protocols

Installing a cluster

To install a cluster or deploy protocols with the IBM Spectrum Scale Installation Toolkit, run the following command:

```
/usr/lpp/mmfs/5.1.2.1/ansible-toolkit/spectrumscale -h
```

To install a cluster manually, use the GPFS packages that are in the following location:

```
/usr/lpp/mmfs/5.1.2.1/gpfs_<rpms/debs>
```

Upgrading a cluster

Complete the following steps to upgrade a cluster by using the IBM Spectrum Scale Installation Toolkit:

1. Review and update the configuration:

```
/usr/lpp/mmfs/5.1.2.1/ansible-toolkit/spectrumscale config update
```

2. Update the cluster configuration to reflect the current cluster configuration:

```
/usr/lpp/mmfs/5.1.2.1/ansible-toolkit/spectrumscale config populate -N <node>
```

3. Run the upgrade:

```
/usr/lpp/mmfs/5.1.2.1/ansible-toolkit/spectrumscale upgrade -h
```

Adding nodes to a cluster

Complete the following steps to add nodes to a cluster by using the IBM Spectrum Scale Installation Toolkit:

1. Add nodes to the cluster definition file:

```
/usr/lpp/mmfs/5.1.2.1/ansible-toolkit/spectrumscale node add -h
```

2. Install IBM Spectrum Scale on the new nodes:

```
/usr/lpp/mmfs/5.1.2.1/ansible-toolkit/spectrumscale install -h
```

3. Deploy protocols on the new nodes:

```
/usr/lpp/mmfs/5.1.2.1/ansible-toolkit/spectrumscale deploy -h
```

Adding NSD or file systems to a cluster:

Complete the following steps to add Network Shared Disks (NSDs) or file systems to a cluster by using the IBM Spectrum Scale Installation Toolkit:

1. Add NSDs or file systems to the cluster definition:

```
/usr/lpp/mmfs/5.1.2.1/ansible-toolkit/spectrumscale nsd add -h
```

2. Install the NSDs or file systems:

```
/usr/lpp/mmfs/5.1.2.1/ansible-toolkit/spectrumscale install -h
```

Updating the cluster definition

To update the cluster definition to reflect the current cluster configuration, run the following command:

```
/usr/lpp/mmfs/5.1.2.1/ansible-toolkit/spectrumscale config populate -N <node>
```

Note: For <node>, pick a node in the cluster for the toolkit to use for automatic configuration.

For more information, see [IBM Spectrum Scale Protocols Quick Overview](#).

Kernel-devel installed version verification

Ensure that installed Kernel-devel versions are compatible with Linux Kernel.

Run the following commands to verify whether the Kernel-devel versions that are installed in the rdbkkvm2 and rdbkkvm3 virtual machine instances are the same as the version that is running in the Linux Kernel:

- ▶ For rdbkkvm2:

```
# uname -a
Linux rdbkkvm2.az12.dal.cpc.ibm.com 4.18.0-348.2.1.el8_5.s390x #1 SMP Mon Nov 8
10:08:25 EST 2021 s390x s390x s390x GNU/Linux

# rpm -q kernel-devel
kernel-devel-4.18.0-348.2.1.el8_5.s390x
```

- ▶ For rdbkkvm3:

```
# uname -a
Linux rdbkkvm3.az12.dal.cpc.ibm.com 4.18.0-348.2.1.el8_5.s390x #1 SMP Mon Nov 8
10:08:25 EST 2021 s390x s390x s390x GNU/Linux

# rpm -q kernel-devel
kernel-devel-4.18.0-348.2.1.el8_5.s390x
```

7.4.2 Configuring IBM Spectrum Scale

Run the command that is shown in Example 7-2 to install the RPMs.

Example 7-2 Installing the RPMs

```
# cd /usr/lpp/mmfs/5.1.2.1/gpfs_rpms
# rpm -ivh
/usr/lpp/mmfs/5.1.2.1/zimon_rpms/rhel8/gpfs.gss.pmcollector-5.1.2-1.el8.s390x.rpm
# rpm -ivh gpfs*rpm
```

7.5 Building the GPFS portability layer

Before starting GPFS, you must build and install the GPFS portability layer. The GPFS portability layer is a loadable kernel module that allows the GPFS daemon to interact with the operating system.

Run the following command to install GPFS portability layer:

```
/usr/lpp/mmfs/bin/mmbuildgpl
```

The output that is shown in Example 7-3 is displayed.

Example 7-3 Installing GPFS portability layer command output

```
-----  
mmbuildgpl: Building GPL (5.1.2.1) module begins at Fri Nov 26 12:24:29 UTC 2021.  
-----  
Verifying Kernel Header...  
  kernel version = 41800348 (418000348002001, 4.18.0-348.2.1.el8_5.s390x,  
4.18.0-348.2.1)  
  module include dir = /lib/modules/4.18.0-348.2.1.el8_5.s390x/build/include  
  module build dir   = /lib/modules/4.18.0-348.2.1.el8_5.s390x/build  
  kernel source dir  = /usr/src/linux-4.18.0-348.2.1.el8_5.s390x/include  
  Found valid kernel header file under  
/usr/src/kernels/4.18.0-348.2.1.el8_5.s390x/include  
Getting Kernel Cipher mode...  
  Will use blkcipher routines  
Verifying Compiler...  
  make is present at /bin/make  
  cpp is present at /bin/cpp  
  gcc is present at /bin/gcc  
  g++ is present at /bin/g++  
  ld is present at /bin/ld  
Verifying Additional System Headers...  
  Verifying kernel-headers is installed ...  
  Command: /bin/rpm -q kernel-headers  
  The required package kernel-headers is installed  
make World ...  
make InstallImages ...  
-----  
mmbuildgpl: Building GPL module completed successfully at Fri Nov 26 12:24:43 UTC  
2021.  
-----
```

7.6 Handling Linux kernel updates

The GPFS portability layer must be rebuilt on every node that undergoes a Linux kernel update. Apply the kernel, restart, and rebuild the GPFS portability layer on each node by using one of the following commands:

- ▶ `/usr/lpp/mmfs/bin/mmbuildgp`
- ▶ `mmchconfig autoBuildGPL=yes`
- ▶ `mmstartup`

The `mmchconfig` command was used in the example that is shown in Example 7-4.

Tip: You can configure a cluster to automatically rebuild the GPL whenever a new level of the Linux kernel or IBM Spectrum Scale are installed. This feature is available on the Linux operating system only. For more information, see this [IBM Documentation web page](#).

Example 7-4 mmchconfig command

```
# /usr/lpp/mmfs/bin/mmchconfig autoBuildGPL=yes
mmchconfig: Command successfully completed
mmchconfig: Propagating the cluster configuration data to all
    affected nodes. This is an asynchronous process.
```

```
# /usr/lpp/mmfs/bin/mm1sconfig
Configuration data for cluster rdbkkvm_cluster1.spectrum:
-----
clusterName rdbkkvm_cluster1.spectrum
clusterId 15364531426110283948
dmapiFileHandleSize 32
minReleaseLevel 5.1.2.0
ccrEnabled yes
cipherList AUTHONLY
sdrNotifyAuthEnabled yes
tiebreakerDisks 3600507630affd05800000000000000003
autoload yes
autoBuildGPL yes
adminMode central
```

```
File systems in cluster rdbkkvm_cluster1.spectrum:
-----
/dev/gpfs1
```

Create a work directory for GPFS configuration:

```
mkdir -p /root/gpfs/
```

Configure Firewall for Spectrum Scale (if local Linux firewall is active):

```
rdbkkvm2:quorum
```

```
rdbkkvm3:quorum
```

Attention: Before you start GPFS cluster configuration, it is critical to create the SSH keys and deploy the public key into `authorized_keys`. The reason is because GPFS uses SSH/SCP to communicate with the cluster nodes.

On `rdbkvm2`

Complete the following steps:

1. Create SSH key pairs using `ssh-keygen`:

```
ssh-keygen -t rsa -b 2048 -N '' -f /root/.ssh/id_rsa.gpfs
```

The output that is shown in Example 7-5 is displayed.

Example 7-5 Output of `ssh-keygen`

```
Generating public/private rsa key pair.
Your identification has been saved in /root/.ssh/id_rsa.gpfs.
Your public key has been saved in /root/.ssh/id_rsa.gpfs.pub.
The key fingerprint is:
SHA256:yCJm3x6WA/f8JfvQ8tilVHLcZmSLFutqFvpGjVVycGU
root@rdbkvm2.az12.dal.cpc.ibm.com
The key's randomart image is:
```

```
+----[RSA 2048]-----+
|
|          ...E|
|          ..+|
|          .+ o|
|      . .   o++|
| + o + S   =+++|
| o o = +   =o= o|
|  . * o * =..|
|    o o #.o   |
|  .   O+=     |
|-----[SHA256]-----+
```

2. Create the `~/.ssh/config` file:

```
cat > ~/.ssh/config << EOF
Host *
    IdentityFile ~/.ssh/id_rsa.gpfs
    StrictHo
stKeyChecking no
EOF
```

3. Confirm whether the `~/.ssh/config` file was created:

```
# cat ~/.ssh/config
Host *
    IdentityFile ~/.ssh/id_rsa.gpfs
    StrictHostKeyChecking no
```

4. Authorize the SSH key:

Note: If the server is not the primary, you must copy `id_rsa` and `id_rsa.pub` to the target server.

```
cat ~/.ssh/id_rsa.gpfs.pub > ~/.ssh/authorized_keys
```

5. Copy the `~/.ssh/` folder to `rdbkkvm3`:

```
scp -r /root/.ssh/ root@rdbkkvm3:/root/
```

6. Update the `/etc/hosts` file so that each node interface that is to be used by GPFS appears only once in the file (highlighted in **bold**):

```
# cat /etc/hosts
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
::1 localhost localhost.localdomain localhost6 localhost6.localdomain6
9.214.220.201 rdbkkvm3
9.214.220.202 rdbkkvm2
```

7. Confirm whether SSH is working between the nodes without prompting for a password:

- On `rdbkkvm2`:

```
# ssh root@rdbkkvm3 date
Fri Nov 26 12:52:05 UTC
```

- On `rdbkkvm3`:

```
# ssh root@rdbkkvm2 date
Fri Nov 26 12:58:47 UTC 2021
```

7.7 GPFS general configuration

Run the following command to create a GPFS cluster from a set of nodes by using `nodes.txt`:

```
/usr/lpp/mmfs/bin/mmcrccluster -N /root/gpfs/nodes.txt --ccr-enable -r /usr/bin/ssh
-R /usr/bin/scp -C rdbkkvm_cluster1.spectrum
```

Note: The `-C` parameter specifies the cluster name. As a convention, we defined our cluster name as: `rdbkkvm_cluster1`.

The following output that is shown in Example 7-6 is produced.

Example 7-6 Output of command to create a GPFS cluster

```
mmcrccluster: Performing preliminary node verification ...
mmcrccluster: Processing quorum and other critical nodes ...
mmcrccluster: Finalizing the cluster data structures ...
mmcrccluster: Command successfully completed
mmcrccluster: Warning: Not all nodes have proper GPFS license designations.
    Use the mmchlicense command to designate licenses as needed.
mmcrccluster: Propagating the cluster configuration data to all
    affected nodes. This is an asynchronous process.
```

Note: When installing the Licensing, the `-N` parameter requires the list of the GPFS nodes, which are separated by a comma.

7.7.1 Installing the licensing

Run the command that is shown in Example 7-7 to install the licensing.

Example 7-7 Installing licensing command

```
# /usr/lpp/mmfs/bin/mmchlicense server --accept -N rdbkkvm2,rdbkkvm3
```

The following nodes will be designated as possessing server licenses:

```
rdbkkvm2
rdbkkvm3
```

```
mmchlicense: Command successfully completed
```

```
mmchlicense: Propagating the cluster configuration data to all
affected nodes. This is an asynchronous process.
```

7.7.2 Validating or listing the cluster configuration

Run the command that is shown in Example 7-8 to validate or list the cluster configuration.

Example 7-8 Validating or listing the cluster configuration

```
# /usr/lpp/mmfs/bin/mmlscluster
```

```
GPFS cluster information
```

```
=====
```

```
GPFS cluster name:      rdbkkvm_cluster1.spectrum
GPFS cluster id:       15364531426110283948
GPFS UID domain:      rdbkkvm_cluster1.spectrum
Remote shell command:  /usr/bin/ssh
Remote file copy command: /usr/bin/scp
Repository type:      CCR
```

Node	Daemon node name	IP address	Admin node name	Designation
1	rdbkkvm2	9.214.220.202	rdbkkvm2	quorum
2	rdbkkvm3	9.214.220.201	rdbkkvm3	quorum

7.7.3 Displaying the state of GPFS cluster

Run the command that is shown in Example 7-9 to display the state of the cluster.

Example 7-9 Displaying the cluster state

```
# /usr/lpp/mmfs/bin/mmgetstate -a
```

```
Node number  Node name  GPFS state
-----
          1  rdbkkvm2  down
          2  rdbkkvm3  down
```

```
# /usr/lpp/mmfs/bin/mmgetstate -N rdbkkvm2,rdbkkvm3
```

```
Node number  Node name  GPFS state
-----
          1  rdbkkvm2  down
          2  rdbkkvm3  down
```

7.7.4 Changing the range ports that are used for command execution

Run the following command to change the range ports that are used for command execution:

```
mmchconfig tscCmdPortRange=60000-61000
```

7.7.5 Configuring FCP Channels to all KVM Compute (GPFS cluster) servers

In our environment, the IBM Z server was defined with FCP F000/F100/F200/F300 addresses. After removing them from the Driver Blacklist, map all LUNS and remove them from LVM in sequence.

Removing FXXX FCP Channels from the driver blacklist

Run the command that is shown in Example 7-10 to remove FXXX FCP channels from the driver blacklist.

Example 7-10 Removing FXXX FCP channels from driver blacklist

```
# cio_ignore -r f000
# cio_ignore -r f100
# cio_ignore -r f200
# cio_ignore -r f300
```

```
# Enable FXXX FTP Channels:
```

```
# chzdev -e zfcpc-host f000
FCP device 0.0.f000 configured
```

```
# chzdev -e zfcpc-host f100
FCP device 0.0.f100 configured
```

```
# chzdev -e zfcplib-host f200
FCP device 0.0.f200 configured
```

```
# chzdev -e zfcplib-host f300
FCP device 0.0.f300 configured
```

The sequence that is shown in Example 7-11 maps LUNs, makes them persistent, removes physical volumes, and validates the configuration.

Example 7-11 Mapping and persisting LUNs, removing physical volumes, and validating configuration

```
# lslns
Scanning for LUNs on adapter 0.0.f000
  at port 0x500507630a399058:
    0x4000400300000000
    0x4000400400000000
    0x4000400500000000
    0x4001400300000000
    0x4001400400000000
    0x4001400500000000
  at port 0x500507630a3c9058:
    0x4000400300000000
    0x4000400400000000
    0x4000400500000000
    0x4001400300000000
    0x4001400400000000
    0x4001400500000000
Scanning for LUNs on adapter 0.0.f100
  at port 0x500507630a319058:
    0x4000400300000000
    0x4000400400000000
    0x4000400500000000
    0x4001400300000000
    0x4001400400000000
    0x4001400500000000
  at port 0x500507630a349058:
    0x4000400300000000
    0x4000400400000000
    0x4000400500000000
    0x4001400300000000
    0x4001400400000000
    0x4001400500000000
Scanning for LUNs on adapter 0.0.f200
  at port 0x500507630a099058:
    0x4000400300000000
    0x4000400400000000
```

```

0x4000400500000000
0x4001400300000000
0x4001400400000000
0x4001400500000000
at port 0x500507630a1c9058:
0x4000400300000000
0x4000400400000000
0x4000400500000000
0x4001400300000000
0x4001400400000000
0x4001400500000000
Scanning for LUNs on adapter 0.0.f300
at port 0x500507630a019058:
0x4000400300000000
0x4000400400000000
0x4000400500000000
0x4001400300000000
0x4001400400000000
0x4001400500000000
at port 0x500507630a149058:
0x4000400300000000
0x4000400400000000
0x4000400500000000
0x4001400300000000
0x4001400400000000
0x4001400500000000

```

Enabling disk paths

Run the commands that are shown in Example 7-12 to enable each disk path.

Example 7-12 Commands used to enable each disk path

For f000:

```

chzdev -e zfcplun 0.0.f000:0x500507630a399058:0x4000400300000000
chzdev -e zfcplun 0.0.f000:0x500507630a399058:0x4000400400000000
chzdev -e zfcplun 0.0.f000:0x500507630a399058:0x4000400500000000
chzdev -e zfcplun 0.0.f000:0x500507630a399058:0x4001400300000000
chzdev -e zfcplun 0.0.f000:0x500507630a399058:0x4001400400000000
chzdev -e zfcplun 0.0.f000:0x500507630a399058:0x4001400500000000

```

```

chzdev -e zfcplun 0.0.f000:0x500507630a3c9058:0x4000400300000000
chzdev -e zfcplun 0.0.f000:0x500507630a3c9058:0x4000400400000000
chzdev -e zfcplun 0.0.f000:0x500507630a3c9058:0x4000400500000000
chzdev -e zfcplun 0.0.f000:0x500507630a3c9058:0x4001400300000000
chzdev -e zfcplun 0.0.f000:0x500507630a3c9058:0x4001400400000000
chzdev -e zfcplun 0.0.f000:0x500507630a3c9058:0x4001400500000000

```

For f100:


```
chzdev -e zfcplun 0.0.f100:0x500507630a319058:0x4000400300000000
chzdev -e zfcplun 0.0.f100:0x500507630a319058:0x4000400400000000
chzdev -e zfcplun 0.0.f100:0x500507630a319058:0x4000400500000000
chzdev -e zfcplun 0.0.f100:0x500507630a319058:0x4001400300000000
chzdev -e zfcplun 0.0.f100:0x500507630a319058:0x4001400400000000
chzdev -e zfcplun 0.0.f100:0x500507630a319058:0x4001400500000000
```

```
chzdev -e zfcplun 0.0.f100:0x500507630a349058:0x4000400300000000
chzdev -e zfcplun 0.0.f100:0x500507630a349058:0x4000400400000000
chzdev -e zfcplun 0.0.f100:0x500507630a349058:0x4000400500000000
chzdev -e zfcplun 0.0.f100:0x500507630a349058:0x4001400300000000
chzdev -e zfcplun 0.0.f100:0x500507630a349058:0x4001400400000000
chzdev -e zfcplun 0.0.f100:0x500507630a349058:0x4001400500000000
```

For f200:

```
chzdev -e zfcplun 0.0.f200:0x500507630a099058:0x4000400300000000
chzdev -e zfcplun 0.0.f200:0x500507630a099058:0x4000400400000000
chzdev -e zfcplun 0.0.f200:0x500507630a099058:0x4000400500000000
chzdev -e zfcplun 0.0.f200:0x500507630a099058:0x4001400300000000
chzdev -e zfcplun 0.0.f200:0x500507630a099058:0x4001400400000000
chzdev -e zfcplun 0.0.f200:0x500507630a099058:0x4001400500000000
```

```
chzdev -e zfcplun 0.0.f200:0x500507630a1c9058:0x4000400300000000
chzdev -e zfcplun 0.0.f200:0x500507630a1c9058:0x4000400400000000
chzdev -e zfcplun 0.0.f200:0x500507630a1c9058:0x4000400500000000
chzdev -e zfcplun 0.0.f200:0x500507630a1c9058:0x4001400300000000
chzdev -e zfcplun 0.0.f200:0x500507630a1c9058:0x4001400400000000
chzdev -e zfcplun 0.0.f200:0x500507630a1c9058:0x4001400500000000
```

For f300:

```
chzdev -e zfcplun 0.0.f300:0x500507630a019058:0x4000400300000000
chzdev -e zfcplun 0.0.f300:0x500507630a019058:0x4000400400000000
chzdev -e zfcplun 0.0.f300:0x500507630a019058:0x4000400500000000
chzdev -e zfcplun 0.0.f300:0x500507630a019058:0x4001400300000000
chzdev -e zfcplun 0.0.f300:0x500507630a019058:0x4001400400000000
chzdev -e zfcplun 0.0.f300:0x500507630a019058:0x4001400500000000
```

```
chzdev -e zfcplun 0.0.f300:0x500507630a149058:0x4000400300000000
chzdev -e zfcplun 0.0.f300:0x500507630a149058:0x4000400400000000
chzdev -e zfcplun 0.0.f300:0x500507630a149058:0x4000400500000000
chzdev -e zfcplun 0.0.f300:0x500507630a149058:0x4001400300000000
chzdev -e zfcplun 0.0.f300:0x500507630a149058:0x4001400400000000
chzdev -e zfcplun 0.0.f300:0x500507630a149058:0x4001400500000000
```

Enabling multipath by using the mpathconf command

Run the following command to enable multipathing:

```
# /sbin/mpathconf --enable
# multipath
# multipath -ll
```

The output that is shown in Example 7-13 is displayed.

Example 7-13 Output of mpathconf command

```
pathe (3600507630affd058000000000000005) dm-12 IBM,2107900
size=256G features='1 queue_if_no_path' hwhandler='1 alua' wp=rw
`-+- policy='service-time 0' prio=50 status=enabled
  |- 0:0:0:1074085888 sdc 8:32 active ready running
  |- 0:0:1:1074085888 sdi 8:128 active ready running
  |- 1:0:0:1074085888 sdu 65:64 active ready running
  |- 1:0:1:1074085888 sdo 8:224 active ready running
  |- 2:0:0:1074085888 sdaa 65:160 active ready running
  |- 2:0:1:1074085888 sdag 66:0 active ready running
  |- 3:0:0:1074085888 sdam 66:96 active ready running
  `-- 3:0:1:1074085888 sdas 66:192 active ready running
mpathd (3600507630affd0580000000000000104) dm-11 IBM,2107900
size=256G features='1 queue_if_no_path' hwhandler='1 alua' wp=rw
`-+- policy='service-time 0' prio=50 status=enabled
  |- 0:0:0:1074020353 sde 8:64 active ready running
  |- 0:0:1:1074020353 sdk 8:160 active ready running
  |- 1:0:0:1074020353 sdw 65:96 active ready running
  |- 1:0:1:1074020353 sdq 65:0 active ready running
  |- 2:0:0:1074020353 sdac 65:192 active ready running
  |- 2:0:1:1074020353 sdai 66:32 active ready running
  |- 3:0:0:1074020353 sdao 66:128 active ready running
  `-- 3:0:1:1074020353 sdau 66:224 active ready running
mpathc (3600507630affd0580000000000000004) dm-10 IBM,2107900
size=256G features='1 queue_if_no_path' hwhandler='1 alua' wp=rw
`-+- policy='service-time 0' prio=50 status=enabled
  |- 0:0:0:1074020352 sdb 8:16 active ready running
  |- 0:0:1:1074020352 sdh 8:112 active ready running
  |- 1:0:0:1074020352 sdt 65:48 active ready running
  |- 1:0:1:1074020352 sdn 8:208 active ready running
  |- 2:0:0:1074020352 sdz 65:144 active ready running
  |- 2:0:1:1074020352 sdaf 65:240 active ready running
  |- 3:0:0:1074020352 sda1 66:80 active ready running
  `-- 3:0:1:1074020352 sdar 66:176 active ready running
mpathb (3600507630affd0580000000000000103) dm-9 IBM,2107900
size=256G features='1 queue_if_no_path' hwhandler='1 alua' wp=rw
`-+- policy='service-time 0' prio=50 status=enabled
  |- 0:0:0:1073954817 sdd 8:48 active ready running
```

```

|- 0:0:1:1073954817 sdj 8:144 active ready running
|- 1:0:0:1073954817 sdv 65:80 active ready running
|- 1:0:1:1073954817 sdp 8:240 active ready running
|- 2:0:0:1073954817 sdab 65:176 active ready running
|- 2:0:1:1073954817 sdah 66:16 active ready running
|- 3:0:0:1073954817 sdan 66:112 active ready running
~- 3:0:1:1073954817 sdat 66:208 active ready running
mpatha (3600507630affd058000000000000003) dm-8 IBM,2107900
size=256G features='1 queue_if_no_path' hwhandler='1 alua' wp=rw
~+~ policy='service-time 0' prio=50 status=active
|- 0:0:0:1073954816 sda 8:0 active ready running
|- 0:0:1:1073954816 sdg 8:96 active ready running
|- 1:0:0:1073954816 sds 65:32 active ready running
|- 1:0:1:1073954816 sdm 8:192 active ready running
|- 2:0:0:1073954816 sdy 65:128 active ready running
|- 2:0:1:1073954816 sdae 65:224 active ready running
|- 3:0:0:1073954816 sdak 66:64 active ready running
~- 3:0:1:1073954816 sdaq 66:160 active ready running
mpathf (3600507630affd0580000000000000105) dm-13 IBM,2107900
size=256G features='1 queue_if_no_path' hwhandler='1 alua' wp=rw
~+~ policy='service-time 0' prio=50 status=enabled
|- 0:0:0:1074085889 sdf 8:80 active ready running
|- 0:0:1:1074085889 sdl 8:176 active ready running
|- 1:0:0:1074085889 sdx 65:112 active ready running
|- 1:0:1:1074085889 sdr 65:16 active ready running
|- 2:0:0:1074085889 sdad 65:208 active ready running
|- 2:0:1:1074085889 sdaj 66:48 active ready running
|- 3:0:0:1074085889 sdap 66:144 active ready running
~- 3:0:1:1074085889 sdav 66:240 active ready running
[root@rdbkvm2 ~]# multipath -ll
mpathe (3600507630affd0580000000000000005) dm-12 IBM,2107900
size=256G features='1 queue_if_no_path' hwhandler='1 alua' wp=rw
~+~ policy='service-time 0' prio=50 status=enabled
|- 0:0:0:1074085888 sdc 8:32 active ready running
|- 0:0:1:1074085888 sdi 8:128 active ready running
|- 1:0:0:1074085888 sdu 65:64 active ready running
|- 1:0:1:1074085888 sdo 8:224 active ready running
|- 2:0:0:1074085888 sdaa 65:160 active ready running
|- 2:0:1:1074085888 sdag 66:0 active ready running
|- 3:0:0:1074085888 sdam 66:96 active ready running
~- 3:0:1:1074085888 sdas 66:192 active ready running
mpathd (3600507630affd0580000000000000104) dm-11 IBM,2107900
size=256G features='1 queue_if_no_path' hwhandler='1 alua' wp=rw
~+~ policy='service-time 0' prio=50 status=enabled
|- 0:0:0:1074020353 sde 8:64 active ready running
|- 0:0:1:1074020353 sdk 8:160 active ready running

```

```

|- 1:0:0:1074020353 sdw 65:96 active ready running
|- 1:0:1:1074020353 sdq 65:0 active ready running
|- 2:0:0:1074020353 sdac 65:192 active ready running
|- 2:0:1:1074020353 sdai 66:32 active ready running
|- 3:0:0:1074020353 sdao 66:128 active ready running
~- 3:0:1:1074020353 sdau 66:224 active ready running
mpathc (3600507630affd058000000000000004) dm-10 IBM,2107900
size=256G features='1 queue_if_no_path' hwhandler='1 alua' wp=rw
~+~ policy='service-time 0' prio=50 status=enabled
|- 0:0:0:1074020352 sdb 8:16 active ready running
|- 0:0:1:1074020352 sdh 8:112 active ready running
|- 1:0:0:1074020352 sdt 65:48 active ready running
|- 1:0:1:1074020352 sdn 8:208 active ready running
|- 2:0:0:1074020352 sdz 65:144 active ready running
|- 2:0:1:1074020352 sdaf 65:240 active ready running
|- 3:0:0:1074020352 sda1 66:80 active ready running
~- 3:0:1:1074020352 sdar 66:176 active ready running
mpathb (3600507630affd0580000000000000103) dm-9 IBM,2107900
size=256G features='1 queue_if_no_path' hwhandler='1 alua' wp=rw
~+~ policy='service-time 0' prio=50 status=enabled
|- 0:0:0:1073954817 sdd 8:48 active ready running
|- 0:0:1:1073954817 sdj 8:144 active ready running
|- 1:0:0:1073954817 sdv 65:80 active ready running
|- 1:0:1:1073954817 sdp 8:240 active ready running
|- 2:0:0:1073954817 sdab 65:176 active ready running
|- 2:0:1:1073954817 sdah 66:16 active ready running
|- 3:0:0:1073954817 sdan 66:112 active ready running
~- 3:0:1:1073954817 sdat 66:208 active ready running
mpatha (3600507630affd0580000000000000003) dm-8 IBM,2107900
size=256G features='1 queue_if_no_path' hwhandler='1 alua' wp=rw
~+~ policy='service-time 0' prio=50 status=active
|- 0:0:0:1073954816 sda 8:0 active ready running
|- 0:0:1:1073954816 sdg 8:96 active ready running
|- 1:0:0:1073954816 sds 65:32 active ready running
|- 1:0:1:1073954816 sdm 8:192 active ready running
|- 2:0:0:1073954816 sdy 65:128 active ready running
|- 2:0:1:1073954816 sdae 65:224 active ready running
|- 3:0:0:1073954816 sdak 66:64 active ready running
~- 3:0:1:1073954816 sdaq 66:160 active ready running
mpathf (3600507630affd0580000000000000105) dm-13 IBM,2107900
size=256G features='1 queue_if_no_path' hwhandler='1 alua' wp=rw
~+~ policy='service-time 0' prio=50 status=enabled
|- 0:0:0:1074085889 sdf 8:80 active ready running
|- 0:0:1:1074085889 sdl 8:176 active ready running
|- 1:0:0:1074085889 sdx 65:112 active ready running
|- 1:0:1:1074085889 sdr 65:16 active ready running

```

```

|- 2:0:0:1074085889 sdad 65:208 active ready running
|- 2:0:1:1074085889 sdaj 66:48 active ready running
|- 3:0:0:1074085889 sdap 66:144 active ready running
^- 3:0:1:1074085889 sdav 66:240 active ready running

```

Note: The multipath commands that are shown in Example 7-13 on page 234 on the other node.

Configuring the Spectrum Network Shared Disk

For our Network Shared Disk (NSD), we use the nomenclature that is in `/dev/disk/by-id/`.

The following Device Names were used for our environment:

```

# ls -la /dev/disk/by-id/scsi-*
lrwxrwxrwx. 1 root root 10 Nov 26 18:58
/dev/disk/by-id/scsi-3600507630affd0580000000000000003 -> ../../dm-8
lrwxrwxrwx. 1 root root 11 Nov 26 18:58
/dev/disk/by-id/scsi-3600507630affd0580000000000000004 -> ../../dm-10
lrwxrwxrwx. 1 root root 11 Nov 26 18:58
/dev/disk/by-id/scsi-3600507630affd0580000000000000005 -> ../../dm-12
lrwxrwxrwx. 1 root root 10 Nov 26 18:58
/dev/disk/by-id/scsi-3600507630affd0580000000000000103 -> ../../dm-9
lrwxrwxrwx. 1 root root 11 Nov 26 18:58
/dev/disk/by-id/scsi-3600507630affd0580000000000000104 -> ../../dm-11
lrwxrwxrwx. 1 root root 11 Nov 26 18:58
/dev/disk/by-id/scsi-3600507630affd0580000000000000105 -> ../../dm-13

/dev/disk/by-id/scsi-3600507630affd0580000000000000003
/dev/disk/by-id/scsi-3600507630affd0580000000000000004
/dev/disk/by-id/scsi-3600507630affd0580000000000000005
/dev/disk/by-id/scsi-3600507630affd0580000000000000103
/dev/disk/by-id/scsi-3600507630affd0580000000000000104
/dev/disk/by-id/scsi-3600507630affd0580000000000000105

```

Storage Area Network

Update the following command with the device names that are found in the Device Names output that is shown in the previous section. The NSD name cannot include the folder path for the device, as shown in Example 7-14.

Example 7-14 SAN

```

cat > ~/gpfs/disk.nsd << EOF
%nsd:
device=/dev/disk/by-id/scsi-3600507630affd0580000000000000003
nsd=3600507630affd0580000000000000003
usage=dataAndMetadata

%nsd:
device=/dev/disk/by-id/scsi-3600507630affd0580000000000000004

```

```
nsd=3600507630affd058000000000000004  
usage=dataAndMetadata
```

```
%nsd:  
device=/dev/disk/by-id/scsi-3600507630affd058000000000000005  
nsd=3600507630affd058000000000000005  
usage=dataAndMetadata
```

```
%nsd:  
device=/dev/disk/by-id/scsi-3600507630affd0580000000000000103  
nsd=3600507630affd0580000000000000103  
usage=dataAndMetadata
```

```
%nsd:  
device=/dev/disk/by-id/scsi-3600507630affd0580000000000000104  
nsd=3600507630affd0580000000000000104  
usage=dataAndMetadata
```

```
%nsd:  
device=/dev/disk/by-id/scsi-3600507630affd0580000000000000105  
nsd=3600507630affd0580000000000000105  
usage=dataAndMetadata  
EOF
```

Confirm whether the file was created by using the following command:

```
# ~/gpfs/disk.nsd
```

Updating VMALLOC kernel values

Before starting GPFS, complete the following steps on each Linux on Z node:

1. Use the `grubby` utility to add `vmalloc=4096G` for all boot entries on your node, as shown in the following example:

```
grubby --update-kernel=ALL --args="vmalloc=4096G"
```

2. Run the `zip1` command.
3. Restart the node.

For more information, see this [IBM Documentation web page](#).

Creating NSD resources

On rdbkkvm2, run the following command to create the NSD resources:

```
/usr/lpp/mmfs/bin/mmcnsd -F ~/gpfs/disk.nsd
```

The output that is shown in Example 7-15 is displayed.

Example 7-15 Output of /usr/lpp/mmfs/bin/mmcnsd -F ~/gpfs/disk.nsd command

```
mmcnsd: Processing disk disk/by-id/scsi-3600507630affd0580000000000000003
mmcnsd: Processing disk disk/by-id/scsi-3600507630affd0580000000000000004
mmcnsd: Processing disk disk/by-id/scsi-3600507630affd0580000000000000005
mmcnsd: Processing disk disk/by-id/scsi-3600507630affd0580000000000000103
mmcnsd: Processing disk disk/by-id/scsi-3600507630affd0580000000000000104
mmcnsd: Processing disk disk/by-id/scsi-3600507630affd0580000000000000105
mmcnsd: Propagating the cluster configuration data to all affected nodes. This
is an asynchronous process.
```

Starting GPFS on all nodes

Run the following command to start GPFS on all nodes:

```
# /usr/lpp/mmfs/bin/mmgetstate -a
```

```
Node number  Node name  GPFS state
-----
           1  rdbkkvm2  down
           2  rdbkkvm3  down
```

```
# /usr/lpp/mmfs/bin/mmstartup -a
```

```
Fri Nov 26 19:28:30 UTC 2021: mmstartup: Starting GPFS ...
```

```
# /usr/lpp/mmfs/bin/mmgetstate -a
```

Note: Wait a few minutes for the process to complete. The cluster takes some time to fully start. In our environment, we waited approximately 5 minutes. If you run the `mmgetstate` command without waiting, you might see the GPFS state as arbitrating.

```
Node number  Node name  GPFS state
-----
           1  rdbkkvm2  arbitrating
           2  rdbkkvm3  arbitrating
```

After a few minutes, the status should change to active:

```
# /usr/lpp/mmfs/bin/mmgetstate -a
```

```
Node number  Node name  GPFS state
-----
           1  rdbkkvm2  active
           2  rdbkkvm3  active
```

7.7.6 Using tiebreaker disks

When running on small GPFS clusters, you might want to have the cluster remain online with only one surviving node.

To achieve this single node, you must add a tiebreaker disk to the quorum configuration. This disk allows you to run with as little as one quorum node available if you can access most of the quorum disks.

Enabling node quorum by using tiebreaker disks starts by designating one or more nodes as quorum nodes (in our case, two quorum nodes are used). Then, 1 - 3 disks are defined as tiebreaker disks by using the `tiebreakerDisks` parameter in the `mmchconfig` command.

In our case, we selected a tiebreaker disk that is part of a GPFS file system.

The following tiebreaker disk was selected:

```
3600507630affd05800000000000000003
```

```
/usr/lpp/mmfs/bin/mmchconfig tiebreakerDisks= 3600507630affd05800000000000000003
```

The following output is shown:

```
mmchconfig: Command successfully completed
mmchconfig: Propagating the cluster configuration data to all
    affected nodes. This is an asynchronous process.
```

Adding tiebreaker disks

Tip: Consider the following points when tiebreaker disks are added:

- ▶ If the tiebreaker disks are part of a file system, GPFS should be running.
- ▶ If the tiebreaker disks are not part of a file system, GPFS can be running or shut down.

You can check whether the tiebreaker disk is set up by using the command that is shown in Example 7-16.

Example 7-16 Checking whether tiebreaker disk is set up

```
# /usr/lpp/mmfs/bin/mmlsconfig
Configuration data for cluster rdbkkvm_cluster1.spectrum:
-----
clusterName rdbkkvm_cluster1.spectrum
clusterId 15364531426110283948
autoload no
dmapiFileHandleSize 32
minReleaseLevel 5.1.2.0
ccrEnabled yes
cipherList AUTHONLY
sdrNotifyAuthEnabled yes
tiebreakerDisks 3600507630affd05800000000000000003
adminMode central
```


File systems in cluster rdbkkvm_cluster1.spectrum:

(none)

7.7.7 Displaying the NSD information

Run the following command to list all NSD resources that were created:

```
# /usr/lpp/mmfs/bin/mm1nsd -X
```

Disk name	NSD volume ID	Device	Devtype	Node name or Class	Remarks
3600507630affd05800000000000000003	DCCA09D661A13374	/dev/dm-8	dmm	rdbkkvm2	
3600507630affd05800000000000000004	DCCA09D661A13375	/dev/dm-11	dmm	rdbkkvm2	
3600507630affd05800000000000000005	DCCA09D661A13376	/dev/dm-15	dmm	rdbkkvm2	
3600507630affd05800000000000000103	DCCA09D661A13377	/dev/dm-9	dmm	rdbkkvm2	
3600507630affd05800000000000000104	DCCA09D661A13379	/dev/dm-13	dmm	rdbkkvm2	
3600507630affd05800000000000000105	DCCA09D661A1337A	/dev/dm-17	dmm	rdbkkvm2	

Run the following command only if you must delete a NSD resource:

```
/usr/lpp/mmfs/bin/mmde1nsd <disk_name>
```

For more information about NSD, see this [IBM Documentation web page](#).

7.8 Working with the General Parallel File System

In this section, we describe working with GPFS in various capacities.

7.8.1 Creating and configuring GPFS

To create and configure GPFS, connect as root on rdbkkvm2 and then, run the following command:

```
/usr/lpp/mmfs/bin/mmcrcfs gpfs1 -F /root/gpfs/disk.nsd -T /gpfs
```

The output that is shown in Example 7-17 is displayed.

Example 7-17 Output of command

The following disks of gpfs1 will be formatted on node rdbkkvm2.az12.dal.cpc.ibm.com:

```
3600507630affd05800000000000000003: size 262144 MB
3600507630affd05800000000000000004: size 262144 MB
3600507630affd05800000000000000005: size 262144 MB
3600507630affd05800000000000000103: size 262144 MB
3600507630affd05800000000000000104: size 262144 MB
3600507630affd05800000000000000105: size 262144 MB
```

Formatting file system ...

Disks up to size 3.06 TB can be added to storage pool system.

Creating Inode File

Creating Allocation Maps

```
Creating Log Files
Clearing Inode Allocation Map
Clearing Block Allocation Map
Formatting Allocation Map for storage pool system
Completed creation of file system /dev/gpfs1.
mmcrfs: Propagating the cluster configuration data to all affected nodes.
This is an asynchronous process.
```

7.8.2 Mounting and validating the GPFS

Run the following command to mount and validate the GPFS:

```
# /usr/lpp/mmfs/bin/mmmount gpfs1 -a
Fri Nov 26 19:47:20 UTC 2021: mmmount: Mounting file systems ...
```

7.8.3 Configuring the SELinux file's context

Note: The fcontext configuration affects the entire cluster.

Run the following command to configure the SELinux file's context:

```
# Configuring SELinux Context Files Permissions:
## add the nfs_t
semanage fcontext -a -t nfs_t "/gpfs(/.*)?"

## restore permissions
restorecon -Rv /gpfs/
```

The output that is shown in Example 7-18 is displayed.

Example 7-18 Configuring SELinux file's context command output

```
Relabeled /gpfs from system_u:object_r:unlabeled_t:s0 to
system_u:object_r:nfs_t:s0
restorecon: Could not set context for /gpfs/.snapshots: Operation not permitted

## validate /gpfs/
ls -alZ /gpfs/

total 257
drwxr-xr-x. 2 root root system_u:object_r:nfs_t:s0      262144 Nov 26 19:46 .
dr-xr-xr-x. 18 root root system_u:object_r:root_t:s0    236 Nov 26 19:46 ..
dr-xr-xr-x. 2 root root system_u:object_r:unlabeled_t:s0 8192 Jan 1 1970
.snapshots

drwxr-xr-x. 7 nova nova system_u:object_r:nfs_t:s0      262144 Nov 4 20:41 .
```

```
## if you want to validate the new SELinux Context, issue command below
semanage fcontext --list |grep ^/gpfs
```

For more information, see this [Red Hat Documentation web page](#).

7.8.4 Starting GPFS automatically

You can specify whether to start the GPFS daemon automatically on a node when it is started.

Whether GPFS automatically starts is determined by using the `autoload` parameter of the `mmchconfig` command. The default setting is to not automatically start GPFS on all nodes.

This setting can be changed by specifying `autoload=yes` with the `mmchconfig` command. This change eliminates the need to start GPFS by running the `mmstartup` command when a node is started.

The `autoload=yes` parameter can be set individually for each node in the cluster. For example, it can be helpful to set `autoload=no` on a node that is undergoing maintenance because operating system upgrades and other software often can require multiple restarts to be completed.

Run the following command to automatically start GPFS:

```
/usr/lpp/mmfs/bin/mmchconfig autoload=yes
```

```
mmchconfig: Command successfully completed
mmchconfig: Propagating the cluster configuration data to all
             affected nodes. This is an asynchronous process.
```

For more information about uninstalling GPFS, see this [IBM Documentation web page](#).

7.8.5 Tiebreaker disk recommendations

During the GPFS installation, we did not allocate a 1 GB disk to be used exclusively for the tiebreaker disk function. It is recommended to reserve this disk space to avoid GDPS from using application disks as tiebreakers.

Therefore, we needed to create a 1 GB disk and assign it to all GPFS nodes. Complete the steps that described in “Adding tiebreaker disks” on page 240 to add this disk to the storage.

Then, run the command that is shown in Example 7-19 on `rdbkvm2` server to replace the tiebreaker disk.

Example 7-19 Replacing the tiebreaker disk

```
/usr/lpp/mmfs/bin/mmchconfig
```

```
clusterId 15364531426110283948
```

```
dmapiFileHandleSize 32
```

```
minReleaseLevel 5.1.2.0
```

```
ccrEnabled yes
```

```
cipherList AUTHONLY
sdrNotifyAuthEnabled yes
tiebreakerDisks 3600507630affd05800000000000000003
autoload yes
autoBuildGPL yes
adminMode central
```

Checking file systems in cluster rdbkvm_cluster1.spectrum

First, enter /dev/gpfs1 to select the rdbkvm_cluster1.spectrum cluster.

Then, issue the **lsluns** command as shown in Example 7-20 to verify whether the new disk is listed in all FCP devices. The new disks are highlighted in **bold**.

Example 7-20 Issuing lsluns

```
# lsluns
Scanning for LUNs on adapter 0.0.f000
  at port 0x500507630a399058:
    0x4000400300000000
    0x4000400400000000
    0x4000400500000000
    0x4001400100000000
    0x4001400300000000
    0x4001400400000000
    0x4001400500000000
  at port 0x500507630a3c9058:
    0x4000400300000000
    0x4000400400000000
    0x4000400500000000
    0x4001400100000000
    0x4001400300000000
    0x4001400400000000
    0x4001400500000000
Scanning for LUNs on adapter 0.0.f100
  at port 0x500507630a319058:
    0x4000400300000000
    0x4000400400000000
    0x4000400500000000
    0x4001400100000000
    0x4001400300000000
    0x4001400400000000
    0x4001400500000000
```

```

at port 0x500507630a349058:
    0x4000400300000000
    0x4000400400000000
    0x4000400500000000
    0x4001400100000000
    0x4001400300000000
    0x4001400400000000
    0x4001400500000000
Scanning for LUNs on adapter 0.0.f200
at port 0x500507630a099058:
    0x4000400300000000
    0x4000400400000000
    0x4000400500000000
    0x4001400100000000
    0x4001400300000000
    0x4001400400000000
    0x4001400500000000
at port 0x500507630a1c9058:
    0x4000400300000000
    0x4000400400000000
    0x4000400500000000
    0x4001400100000000
    0x4001400300000000
    0x4001400400000000
    0x4001400500000000
Scanning for LUNs on adapter 0.0.f300
at port 0x500507630a019058:
    0x4000400300000000
    0x4000400400000000
    0x4000400500000000
    0x4001400100000000
    0x4001400300000000
    0x4001400400000000
    0x4001400500000000
at port 0x500507630a149058:
    0x4000400300000000
    0x4000400400000000
    0x4000400500000000
    0x4001400100000000
    0x4001400300000000
    0x4001400400000000
    0x4001400500000000

```

Run the commands that are shown in Example 7-21 to enable the new disk.

Note: You must run these commands on *all* GPFS nodes to make the new disk available on all FCP devices.

Example 7-21 Enabling the new disk

```
chzdev -e zfcplun 0.0.f000:0x500507630a399058:0x4001400100000000
chzdev -e zfcplun 0.0.f000:0x500507630a3c9058:0x4001400100000000
chzdev -e zfcplun 0.0.f100:0x500507630a319058:0x4001400100000000
chzdev -e zfcplun 0.0.f100:0x500507630a349058:0x4001400100000000
chzdev -e zfcplun 0.0.f200:0x500507630a099058:0x4001400100000000
chzdev -e zfcplun 0.0.f200:0x500507630a1c9058:0x4001400100000000
chzdev -e zfcplun 0.0.f300:0x500507630a019058:0x4001400100000000
chzdev -e zfcplun 0.0.f300:0x500507630a149058:0x4001400100000000
```

Run the `multipath -ll` command to get the multipath Linux device name.

The output is highlighted in **bold** in Example 7-22.

Example 7-22 Output of multipath -ll command

```
mpathe (3600507630affd058000000000000005) dm-15 IBM,2107900
size=256G features='1 queue_if_no_path' hwhandler='1 alua' wp=rw
`-+- policy='service-time 0' prio=50 status=active
  |- 0:0:0:1074085888 sdc 8:32 active ready running
  |- 0:0:1:1074085888 sdi 8:128 active ready running
  |- 1:0:0:1074085888 sdo 8:224 active ready running
  |- 1:0:1:1074085888 sdu 65:64 active ready running
  |- 2:0:0:1074085888 sdaa 65:160 active ready running
  |- 2:0:1:1074085888 sdag 66:0 active ready running
  |- 3:0:0:1074085888 sdam 66:96 active ready running
  `- 3:0:1:1074085888 sdas 66:192 active ready running
mpathd (3600507630affd0580000000000000104) dm-13 IBM,2107900
size=256G features='1 queue_if_no_path' hwhandler='1 alua' wp=rw
`-+- policy='service-time 0' prio=50 status=active
  |- 0:0:0:1074020353 sde 8:64 active ready running
```

```

|- 0:0:1:1074020353 sdk 8:160 active ready running
|- 1:0:0:1074020353 sdq 65:0 active ready running
|- 1:0:1:1074020353 sdw 65:96 active ready running
|- 2:0:0:1074020353 sdac 65:192 active ready running
|- 2:0:1:1074020353 sdai 66:32 active ready running
|- 3:0:0:1074020353 sdao 66:128 active ready running
~- 3:0:1:1074020353 sdau 66:224 active ready running
mpathc (3600507630affd0580000000000000004) dm-11 IBM,2107900
size=256G features='1 queue_if_no_path' hwhandler='1 alua' wp=rw
~+- policy='service-time 0' prio=50 status=active
|- 0:0:0:1074020352 sdb 8:16 active ready running
|- 0:0:1:1074020352 sdh 8:112 active ready running
|- 1:0:0:1074020352 sdn 8:208 active ready running
|- 1:0:1:1074020352 sdt 65:48 active ready running
|- 2:0:0:1074020352 sdz 65:144 active ready running
|- 2:0:1:1074020352 sdaf 65:240 active ready running
|- 3:0:0:1074020352 sda1 66:80 active ready running
~- 3:0:1:1074020352 sdar 66:176 active ready running
mpathb (3600507630affd0580000000000000103) dm-9 IBM,2107900
size=256G features='1 queue_if_no_path' hwhandler='1 alua' wp=rw
~+- policy='service-time 0' prio=50 status=active
|- 0:0:0:1073954817 sdd 8:48 active ready running
|- 0:0:1:1073954817 sdj 8:144 active ready running
|- 1:0:0:1073954817 sdp 8:240 active ready running
|- 1:0:1:1073954817 sdv 65:80 active ready running
|- 2:0:0:1073954817 sdab 65:176 active ready running
|- 2:0:1:1073954817 sdah 66:16 active ready running
|- 3:0:0:1073954817 sdan 66:112 active ready running
~- 3:0:1:1073954817 sdat 66:208 active ready running
mpatha (3600507630affd0580000000000000003) dm-8 IBM,2107900

```

size=256G features='1 queue_if_no_path' hwhandler='1 alua' wp=rw

^-+- policy='service-time 0' prio=50 status=active

| - 0:0:0:1073954816 sda 8:0 active ready running

| - 0:0:1:1073954816 sdg 8:96 active ready running

| - 1:0:0:1073954816 sdm 8:192 active ready running

| - 1:0:1:1073954816 sds 65:32 active ready running

| - 2:0:0:1073954816 sdy 65:128 active ready running

| - 2:0:1:1073954816 sdae 65:224 active ready running

| - 3:0:0:1073954816 sdak 66:64 active ready running

^- 3:0:1:1073954816 sdaq 66:160 active ready running

mpathg (3600507630affd0580000000000000101) dm-20 IBM,2107900

size=1.0G features='1 queue_if_no_path' hwhandler='1 alua' wp=rw

^-+- policy='service-time 0' prio=50 status=active

| - 0:0:1:1073823745 sdaw 67:0 active ready running

| - 0:0:0:1073823745 sdax 67:16 active ready running

| - 1:0:1:1073823745 sday 67:32 active ready running

| - 1:0:0:1073823745 sdaz 67:48 active ready running

| - 2:0:0:1073823745 sdba 67:64 active ready running

| - 2:0:1:1073823745 sdbb 67:80 active ready running

| - 3:0:0:1073823745 sdbc 67:96 active ready running

^- 3:0:1:1073823745 sdbd 67:112 active ready running

mpathf (3600507630affd0580000000000000105) dm-17 IBM,2107900

size=256G features='1 queue_if_no_path' hwhandler='1 alua' wp=rw

^-+- policy='service-time 0' prio=50 status=active

| - 0:0:0:1074085889 sdf 8:80 active ready running

| - 0:0:1:1074085889 sd1 8:176 active ready running

| - 1:0:0:1074085889 sdr 65:16 active ready running

| - 1:0:1:1074085889 sdx 65:112 active ready running

| - 2:0:0:1074085889 sdad 65:208 active ready running

| - 2:0:1:1074085889 sdaj 66:48 active ready running


```
|- 3:0:0:1074085889 sdap 66:144 active ready running
~- 3:0:1:1074085889 sdav 66:240 active ready running
```

7.8.6 Setting up a tiebreaker disk

Complete the following steps to use 3600507630affd058000000000000101 to set up as tiebreaker disk:

1. Create `~/gpfs/tiebreak.nsd` file with the following content:

```
%nsd:
device=/dev/disk/by-id/scsi-3600507630affd058000000000000101
nsd=3600507630affd05800000000000000101
usage=dataAndMetadata
```

2. Add the new NSD disk into GPFS:

```
/usr/lpp/mmfs/bin/mmcnsd -F ~/gpfs/tiebreak.nsd
```

3. Update GPFS configuration with the new tiebreaker disk:

```
# /usr/lpp/mmfs/bin/mmchconfig
tiebreakerDisks=3600507630affd058000000000000101
mmchconfig: Command successfully completed
mmchconfig: Propagating the cluster configuration data to all
affected nodes. This is an asynchronous process.
```

4. Confirm whether the tiebreaker disk configuration was updated:

```
# /usr/lpp/mmfs/bin/mmlsconfig
Configuration data for cluster rdbkkvm_cluster1.spectrum:
-----
clusterName rdbkkvm_cluster1.spectrum
clusterId 15364531426110283948
dmapiFileHandleSize 32
minReleaseLevel 5.1.2.0
ccrEnabled yes
cipherList AUTHONLY
sdrNotifyAuthEnabled yes
autoload yes
autoBuildGPL yes
tiebreakerDisks 3600507630affd058000000000000101
adminMode central
```

7.8.7 Enabling Persistent Reserve

GPFS can use Persistent Reserve (PR) to improve failover times, but with the following restrictions:

- ▶ PR is supported on AIX and Linux nodes. However, consider the following points:
 - If the disks include defined NSD servers, the NSD server nodes all must be running AIX, or they all must be running Linux.
 - If the disks are SAN-attached to all nodes, the SAN-attached nodes in the cluster all must be running AIX, or they all must be running Linux.
- ▶ The disk subsystems must support PR.

- ▶ GPFS supports a mix of PR disks and other disks. However, you realize improved failover times only if all of the disks in the cluster support PR.
- ▶ GPFS supports only PR in the local cluster. Remote mounts must access the disks through an NSD server.
- ▶ When you enable or disable PR, you must stop GPFS on all nodes.
- ▶ Before enabling PR, ensure that all disks are in the same initial state.

Complete the following steps to enable PR steps:

1. Shutdown GPFS cluster on all nodes:

```
Sat Dec 11 19:14:42 UTC 2021: mmshutdown: Starting force unmount of GPFS file systems
```

```
Sat Dec 11 19:14:47 UTC 2021: mmshutdown: Shutting down GPFS daemons
```

```
Sat Dec 11 19:14:55 UTC 2021: mmshutdown: Finished
```

2. Run the **mmchconfig** command to enable PR:

```
# /usr/lpp/mmfs/bin/mmchconfig usePersistentReserve=yes
```

```
Verifying GPFS is stopped on all nodes ...
```

```
mmchconfig: Processing disk 3600507630affd05800000000000000003
```

```
mmchconfig: Processing disk 3600507630affd05800000000000000004
```

```
mmchconfig: Processing disk 3600507630affd05800000000000000005
```

```
mmchconfig: Processing disk 3600507630affd05800000000000000103
```

```
mmchconfig: Processing disk 3600507630affd05800000000000000104
```

```
mmchconfig: Processing disk 3600507630affd05800000000000000105
```

```
mmchconfig: Processing disk 3600507630affd05800000000000000101
```

```
mmchconfig: Command successfully completed
```

```
mmchconfig: Propagating the cluster configuration data to all affected nodes. This is an asynchronous process.
```

3. Set the recovery times by issuing the following command:

Note: For fast recovery times with PR, you also must set the `failureDetectionTime` configuration parameter with a recommended value is 10.

```
# /usr/lpp/mmfs/bin/mmchconfig failureDetectionTime=10
```

```
Verifying GPFS is stopped on all nodes ...
```

```
mmchconfig: Command successfully completed
```

```
mmchconfig: Propagating the cluster configuration data to all affected nodes. This is an asynchronous process.
```

4. Confirm whether the usePersistentReserve and failure-DetectionTime values were set:

```
# /usr/lpp/mmfs/bin/mmlsconfig
Configuration data for cluster rdbkkvm_cluster1.spectrum:
-----
clusterName rdbkkvm_cluster1.spectrum
clusterId 15364531426110283948
dmapiFileHandleSize 32
minReleaseLevel 5.1.2.0
ccrEnabled yes
cipherList AUTHONLY
sdrNotifyAuthEnabled yes
autoload yes
autoBuildGPL yes
tiebreakerDisks 3600507630affd0580000000000000101
usePersistentReserve yes
failureDetectionTime 10
adminMode central
```

5. Determine whether the disks on the servers and the disks of a specific node have PR enabled by running the following command from the node:

```
# /usr/lpp/mmfs/bin/mmlsnsd -X
```

The system responds with output that is similar to the following example:

Disk name	NSD volume ID	Device	Devtype	Node name or Class	Remarks
3600507630affd058000000000000003	DCCA09D661A13374	/dev/dm-8	dmm	rdbkkvm2	pr=yes
3600507630affd058000000000000004	DCCA09D661A13375	/dev/dm-11	dmm	rdbkkvm2	pr=yes
3600507630affd058000000000000005	DCCA09D661A13376	/dev/dm-15	dmm	rdbkkvm2	pr=yes
3600507630affd0580000000000000101	DCCA09D661B4E6CD	/dev/dm-20	dmm	rdbkkvm2	pr=yes
3600507630affd0580000000000000103	DCCA09D661A13377	/dev/dm-9	dmm	rdbkkvm2	pr=yes
3600507630affd0580000000000000104	DCCA09D661A13379	/dev/dm-13	dmm	rdbkkvm2	pr=yes
3600507630affd0580000000000000105	DCCA09D661A1337A	/dev/dm-17	dmm	rdbkkvm2	pr=yes

Note: If the GPFS daemon was started on all the nodes in the cluster and the file system was mounted on all nodes that have direct access to the disks, **pr=yes** must be set on all hdisks. A problem exists if you do not see the pr=yes flag.

For more information, about PR errors, see this [IBM Documentation web page](#).



Using IBM Secure Execution

This chapter describes the value of IBM Secure Execution, discusses how it works, and contains implementation examples for the reader. It also describes some optional steps that you might want to take to further secure your Kernel-based Virtual Machine (KVM)-based virtual server.

This chapter includes the following topics:

- ▶ “Introduction to IBM Secure Execution” on page 254
- ▶ “How IBM Secure Execution works” on page 255
- ▶ “Enabling and verifying that the CPC is Secure Execution ready” on page 255
- ▶ “KVM host and guest software requirements” on page 256
- ▶ “Enabling an Ubuntu 20.04 LTS KVM host for IBM Secure Execution” on page 257
- ▶ “Enabling an SLES 15 SP2 KVM host for IBM Secure Execution” on page 258
- ▶ “Enabling an Ubuntu 20.04 KVM Guest for IBM Secure Execution” on page 261
- ▶ “Enabling a SLES 15 SP2 KVM Guest for IBM Secure Execution” on page 279
- ▶ “Enabling a RHEL KVM Guest for Secure Execution” on page 304

8.1 Introduction to IBM Secure Execution

One potential inhibitor to cloud adoption is trust. Trust of the cloud service provider and its administrators, software, and procedures. Theft of intellectual capital, customer-sensitive personal information, financial information, or classified data can be devastating.

However, what if virtual servers can be hosted in manner where the hypervisor had no access to any of the virtual server data?

In this case, you do not have to worry about the following issues:

- ▶ Trusting the hypervisor administrator not to correctly access your data
- ▶ That the hypervisor administrator correctly secured the hypervisor environment from malicious insider and outsider threats
- ▶ About a software defect that allows unauthorized access into the hypervisor and a malicious individual stealing data from within your virtual server
- ▶ Accusations of theft by way of the hypervisor if you are the cloud provider because no access exists to the virtual servers data

IBM Secure Execution is designed to protect against the following threats:

- ▶ Guest data theft or corruption by way of malicious operations from the Hardware Management Console (HMC) or Support Element (SE) consoles
- ▶ Guest data theft or corruption by way of malicious operation of the hypervisor through malicious or negligent operation
- ▶ Guest data theft or corruption by way of a hacked or corrupted hypervisor environment

IBM Secure Execution does not protect against the following threats:

- ▶ Damage because of inappropriate physical access and operation
- ▶ Physical theft and inspection of hardware such as memory
- ▶ Denial of service attacks
- ▶ Improper configuration or operation of the guest by the guest administrators
- ▶ Theft of guest information through unsecured I/O channels
- ▶ Loading of malicious code into the guest over the network

8.2 How IBM Secure Execution works

Each IBM LinuxONE III has a key pair for which the private key is accessible only in the IBM LinuxONE hardware and firmware.

A customer can prepare an encrypted and integrity protected Linux image. Consider the following points:

- ▶ The encrypted image can be run on only a virtual machine (VM) in the hosts to which it was built to run.
- ▶ The Linux image cannot be decrypted on any other hosts.
- ▶ The secure guest owner must be sure to encrypt the disk and network communications of the guest. This process can be accomplished by using different methods.

The IBM LinuxONE hardware and firmware; that is, Ultravisor (UV), ensures that unencrypted memory or processor state of a running secure execution guest cannot be accessed by the Linux KVM hypervisor, SEcannot, or HMC.

The IBM LinuxONE UV hardware and firmware detects when memory integrity of a running secure execution guest is violated.

8.3 Enabling and verifying that the CPC is Secure Execution ready

The IBM Secure Execution keys must be installed before IBM Secure Execution can be used. A key bundle must be installed at the time of the machine installation by the IBM service representative.

8.3.1 Importing a key bundle into LinuxONE

To use IBM Secure Execution, you must import the key bundle first. To perform this task, you need service role authorization. If you log in without the service role authorization, you do not see the Update buttons.

Complete the following steps:

1. Log in to the IBM LinuxONE Support Element or HMC.
2. Open **System Details** and select the **Instance Information** tab (see Figure 8-1 on page 256).
3. If the Host key shows as not installed, click **Update**.

Note: You must import the key only once because it persists across a power-on reset.

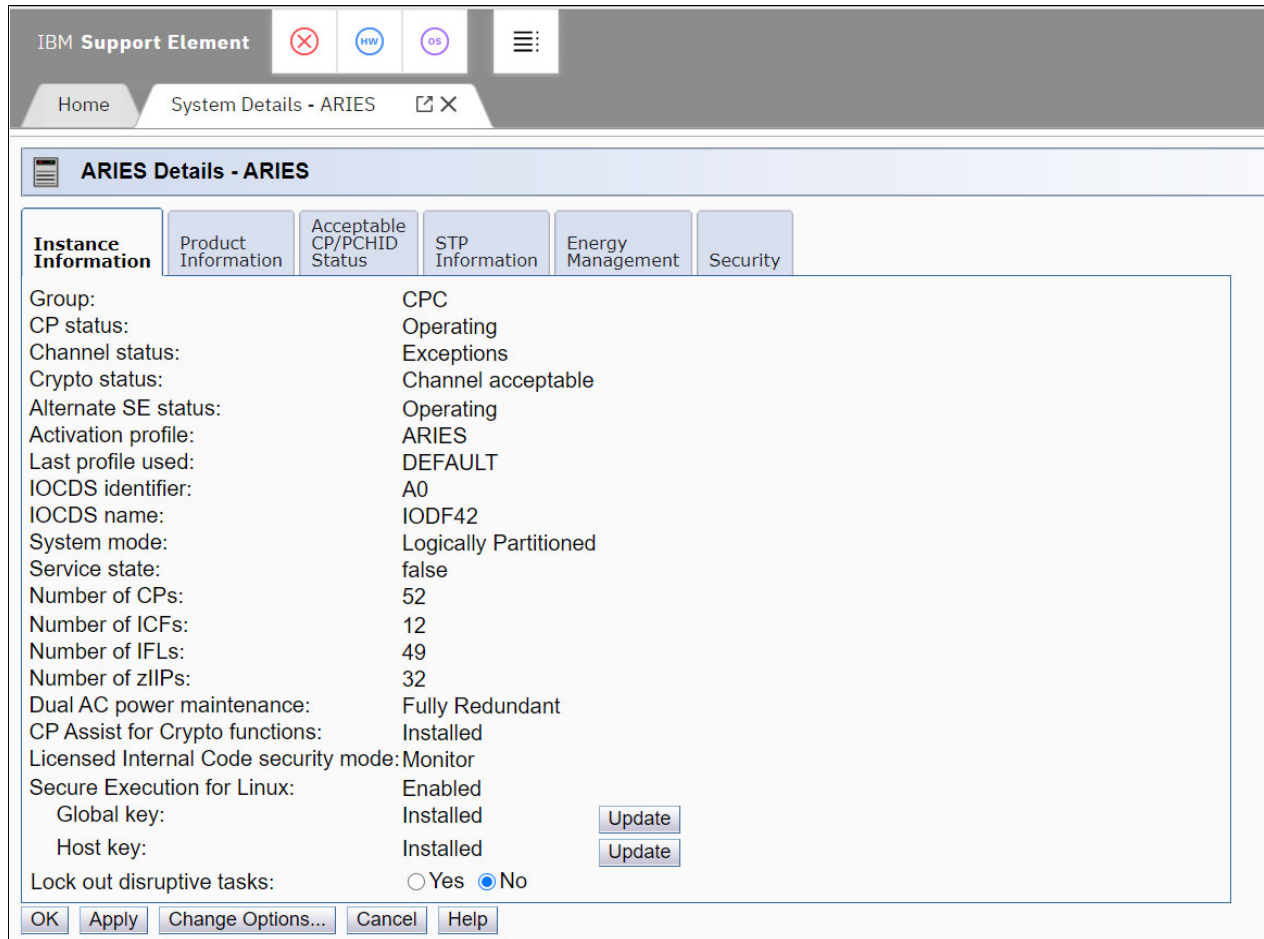


Figure 8-1 CPC Instance information

8.4 KVM host and guest software requirements

In this section, we describe the minimum software requirements for the KVM host and KVM guests.

KVM host software features the following minimum requirements:

- ▶ Ubuntu 20.04 LTS
- ▶ SLES 15 SP2
- ▶ Red Hat Enterprise Linux 8.2

KVM guest software features the following minimum requirements:

- ▶ RHEL 7.8
- ▶ RHEL 8.2
- ▶ SLES 12 SP5
- ▶ SLES 15 SP2
- ▶ Ubuntu 20.04

Because the host is not allowed to access guest memory and state, specific KVM features are not supported, including the following examples:

- ▶ Live migration (offline migration is possible if the guest is built for more than one host).
- ▶ Save to and restore from disk.
- ▶ Hypervisor-initiated memory dump.
- ▶ Pass-through of host devices; for example, PCI, CCW, and crypto AP.
- ▶ The use of large memory pages on the host for backing guest memory.
- ▶ Memory ballooning through a virtio-balloon device.

8.5 Enabling an Ubuntu 20.04 LTS KVM host for IBM Secure Execution

The tasks that are described in this section typically are performed by the KVM host system administrator or cloud provider.

To enable the KVM host for IBM Secure Execution, add the following statement to the kernel parameters line of the `zipl.conf` file:

```
prot_virt=1
```

Complete the following steps to add the parameter to the kernel:

1. Add the kernel parameter to the RHEL host:

```
[root@rdbkvm1 entries]# grubby --update-kernel=ALL --args="prot_virt=1"
[root@rdbkvm1 entries]# dracut -f
```
2. Run the `zipl` command and then, restart your KVM hypervisor.
3. Validate that the kernel parameter addition was successful by checking the contents of `dmesg` for the ultravisor message, as shown in Example 8-1.

Example 8-1 Validating that the kernel parameter change took effect

```
[root@rdbkvm1 ~]# dmesg | grep -i ultra
[ 0.422350] prot_virt: Reserving 578MB as ultravisor base storage
```

You also can check the partition details instance information for the Secure Execution status, as shown in Figure 8-2.

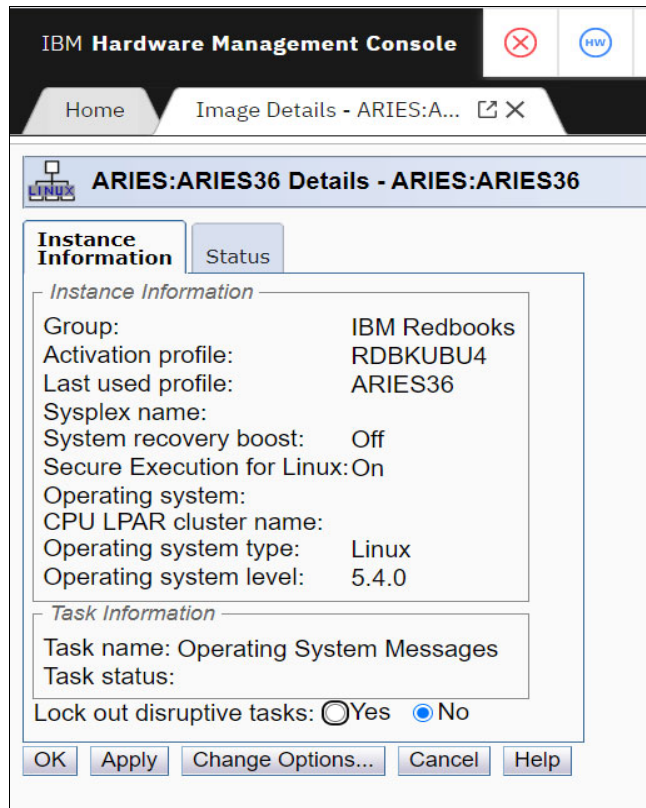


Figure 8-2 Partition Instance information

8.6 Enabling an SLES 15 SP2 KVM host for IBM Secure Execution

The SUSE boot loader process is different than the Ubuntu process. SLES 15 uses two boot loaders: first, it uses **zipl** and then, it uses **grub2**. Two boot loaders require some changes that you might need to make to both boot loaders.

Note: The `zipl.conf` or `grub` configuration file cannot be edited. These files are automatically rebuilt by scripts; therefore, you must update the templates that are provided by SLES 15.

Complete the following steps to enable the SLES 15 host for IBM Secure Execution:

1. To add an entry to the `/etc/grub.d/40_custom` file, complete the following steps:
 - a. Copy the first boot entry from the `/boot/grub2/config/grub.cfg` file and find the line that includes:

```
### BEGIN /etc/grub.d/10_linux ###
```
 - b. Copy the menu entry starting after that line. The contents should be similar to the contents that is shown in Example 8-2 on page 259.

Example 8-2 Menu entry to be copied to the /etc/grub.d/40_custom file

```
menuentry 'SLES 15-SP2' --hotkey=1 --class sles --class gnu-linux --class gnu
--class os $menuentry_id_option
'gnulinux-simple-741778d7-a0d2-4d1b-ad45-5e19f7d9eb39' {
    set gfxpayload=text
    insmod gzio
    echo 'Loading Linux 5.3.18-22-default ...'
    linux ${btrfs_subvol}/boot/image-5.3.18-22-default
root=UUID=741778d7-a0d2-4d1b-ad45-5e19f7d9eb39 ${extra_cmdline} hvc_iucv=8
TERM=dumb mitigations=auto cio_ignore=all,!ipldev,!condev
    echo 'Loading initial ramdisk ...'
    initrd ${btrfs_subvol}/boot/initrd-5.3.18-22-defaul}
```

- c. After you paste the menu entry that is shown in Example 8-2 into /etc/grub.d/40_custom, complete the following required modifications:
- Change the name of the entry in the menu from SLES 15-SP2 to something unique.
 - Update the \$menuentry_id_option to be unique for this new entry.
 - Update the echo message to remove the specific kernel level.
 - Change the Linux and initrd lines to start to the generic symlinks of /boot/image and /boot/initrd, rather than have the specific levels. This change must be made because the grub menu is rebuilt every time that the server is patched and includes new kernel levels.
- d. Add **prot_virt=1** to the kernel parameter line. A modified version is shown in Example 8-3.

Example 8-3 Making the changes to enable Secure Execution

```
!/bin/sh
exec tail -n +3 $0
# This file provides an easy way to add custom menu entries. Simply type the
# menu entries you want to add after this comment. Be careful not to change
# the 'exec tail' line above.

menuentry 'SLES 15-SP2 Ultravisor' --hotkey=5 --class sles --class gnu-linux
--class gnu --class os $menuentry_id_option
'gnulinux-simple-741778d7-a0d2-4d1b-ad45-5e19f7d9eb39ultra' {
    set gfxpayload=text
    insmod gzio
    echo 'Loading Linux 5.x ...'
    linux ${btrfs_subvol}/boot/image
root=UUID=741778d7-a0d2-4d1b-ad45-5e19f7d9eb39 ${extra_cmdline} hvc_iucv=8
TERM=dumb mitigations=auto cio_ignore=all,!ipldev,!condev prot_virt=1
    echo 'Loading initial ramdisk ...'
    initrd ${btrfs_subvol}/boot/initrd}
```

2. After the custom grub entry is completed, run **grub2-mkconfig**, as shown in Example 8-4.

Example 8-4 Generating grub configuration file

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
Generating grub configuration file ...
Found linux image: /boot/image-5.3.18-22-default
Found initrd image: /boot/initrd-5.3.18-22-default
done
```

3. Confirm the grub2 menu entry for the custom configuration. In Example 8-5 you can see `grube2-once --enum` is run to determine the number for the custom grub2 entry (menu entry number 2, in this example).

Example 8-5 Creating a grub2 menu item

```
rdbkkvms:/etc/grub.d # grub2-once --enum
0      SLES 15-SP2
1>0    Advanced options for SLES 15-SP2>SLES 15-SP2, with Linux 5.3.18-22-default
1>1    Advanced options for SLES 15-SP2>SLES 15-SP2, with Linux 5.3.18-22-default
(recovery mode)
2      SLES 15-SP2 Ultravisor
3>0    --hotkey=s>--hotkey=1
3>1    --hotkey=s>--hotkey=2
3>2    --hotkey=s>--hotkey=3
```

4. The KVM host must be IPLed with the new grub kernel parameter. Use the HMC Load parameter to select the `zipl` and grub menu selection. In Figure 8-3, the value of `0G2` is shown; that is, the `zipl` entry 0 and grub entry number is 2.

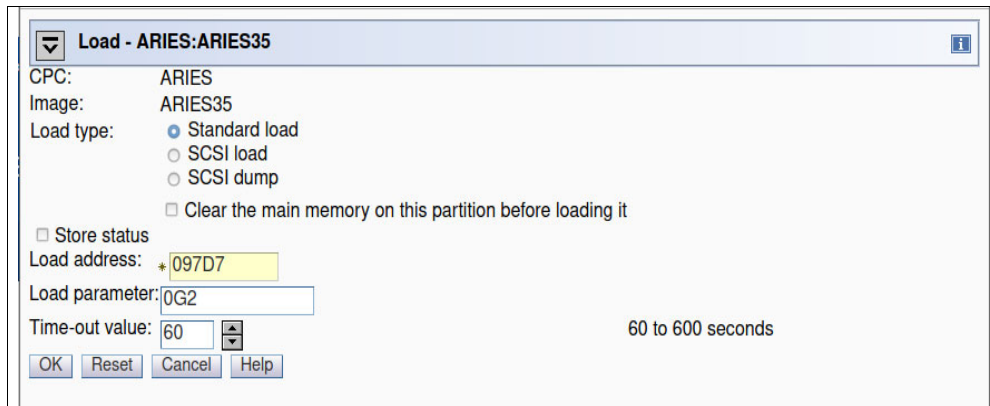


Figure 8-3 HMC Load parameter with grub2 menu selection

After the KVM host is IPLed, validate that the parameter took effect and the IBM Secure Execution Ultravisor is active. In Figure 8-4, you can see that `prot_virt=1` is part of the active kernel parameter and the Ultravisor reserved some memory during start.

```
# cat /proc/cmdline
root=UUID=741778d7-a0d2-4d1b-ad45-5e19f7d9eb39 hvc_iucv=8 TERM=dumb
mitigations=auto cio_ignore=all,!ipldev,!condev prot_virt=1
# dmesg | grep -i ultra
[ 0.338818] prot_virt: Reserving 98MB as ultravisor base storage
#
```

Figure 8-4 Ultravisor verification

8.7 Enabling an Ubuntu 20.04 KVM Guest for IBM Secure Execution

The tasks in this section all are performed by the KVM Guest virtual server administrator. The one exception might that the domain XML for the guest must be edited to include `iommu='on'` on each virtio device. This task is performed by the KVM Host administrator or cloud provider.

Note: We do not use the Pervasive Encryption approach with protected key technology because Crypto AP pass-through is *not* supported by IBM Secure Execution at the time of this writing.

In this example, we perform the following steps:

1. Install a standard Linux guest on encrypted disk storage.
2. Update KVM guest `/etc/crypttab` to avoid entering password at start.
3. Edit the `domain.xml` to include `iommu='on'`.
4. Obtain the host key documents for the CEC.
5. Validate the key material.
6. Build a secured initrd image file by using `genprotimg` on KVM guest.
7. Update guest `zipl` to start with a secured image in Secure Execution mode.
8. Remove any start option that is *not* in Secure Execution mode.
9. Remove unencrypted, older artifacts from `/boot`.

These steps are described next.

8.7.1 Installing a standard Linux guest on encrypted disk storage

We used `virt-install` to deploy a guest in a single command. This process created an 8 Gb `qcow2` image file (see Figure 8-5).

```
virt-install --name secguest1 --memory 4096 --disk size=8 \  
> --cdrom /var/lib/libvirt/images/ubuntu-20.04-legacy-server-s390x.iso
```

Figure 8-5 One line installation command

- ▶ To simplify initial installation, we chose to auto-configure networking by using DHCP (autoconfig networking [DHCP]).
- ▶ To simplify the disk installation, we chose the following Guided selection for disk storage, which uses the entire disk, and setup encrypted Logical Volume Manager (LVM).

We also provided an Encryption Phrase. Store this phrase in a safe place because this phrase is used to decrypt the root file system.

After the installation completes, the server restarts and you must supply the pass phrase when prompted so the root file system is decrypted, as shown in Figure 8-6. This manual process is temporary because we are automating this aspect.

```
Begin: Running /scripts/init-premount ... done.
Begin: Mounting root file system ... Begin: Running /scripts/local-top ...
Volume group "vgsecguest1" not found

Cannot process volume group vgsecguest1

Please unlock disk vda6_crypt:
```

Figure 8-6 Passphrase prompt to decrypt

At this stage, you have a virtual server with the root partition encrypted (/boot is *not* encrypted). While the root partition is encrypted, we are not yet operating in a Secure Execution mode.

Next, we shut down the virtual server and copy the qcow2 file as a backup in case any unrecoverable mistakes were made (see Figure 8-7). After the qcow2 file is copied, we restart the server.

```
root@rdbkubu4:/var/lib/libvirt/images# cp secguest1.qcow2
secguest1.qcow2.backup
root@rdbkubu4:/var/lib/libvirt/images#
```

Figure 8-7 Backup of new virtual server

8.7.2 Updating KVM guest /etc/crypttab to avoid entering a password at start

From a security and operational perspective, we do not want any manual prompts at start to which a user must respond. To address this issue, we created a Linux Unified Key Setup (LUKS) key file and updated /etc/crypttab to use it.

Although these components initially are in /filesystem, you must rebuild initrd/initramfs to include them. With a standard initrd/initramfs, these components are stored in the clear and present a secure risk.

With IBM Secure Execution, this information is encrypted and can be decrypted only in a valid Secure Execution environment.

Figure 8-8 shows that the volume group for root and swap is derived from vda6_crypt.

```
# lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
sr0                                  11:0    1 1024M  0 rom
vda                                  252:0    0    8G  0 disk
├─vda1                               252:1    0   512M  0 part
├─vda2                               252:2    0    1K  0 part
├─vda5                               252:5    0   731M  0 part  /boot
├─vda6                               252:6    0   6.8G  0 part
│   └─vda6_crypt                    253:0    0   6.8G  0 crypt
│       ├─vgsecguest1-root          253:1    0   5.8G  0 lvm  /
│       └─vgsecguest1-swap_1       253:2    0    980M  0 lvm  [SWAP]
```

Figure 8-8 Linux Block devices

With cryptsetup status, we can see that vda6_crypt is a LUKS2 device with aes-xts encryption (see Figure 8-9).

```
# cryptsetup -v status vda6_crypt
/dev/mapper/vda6_crypt is active and is in use.
type:      LUKS2
cipher:    aes-xts-plain64
keysize:   512 bits
key location: keyring
device:    /dev/vda6
sector size: 512
offset:    32768 sectors
size:      14190592 sectors
mode:      read/write
flags:     discards
Command successful.
```

Figure 8-9 cryptsetup status

Complete the following steps:

1. Create a key file that we can use in /etc/crypttab. In Figure 8-10, see that the /etc/luks directory is created, a key file is created, and permissions are set on the directory and file; therefore, only root has read access to this file.

```
# mkdir /etc/luks
root@secguest1:/etc# dd if=/dev/urandom of=/etc/luks/boot_os.keyfile bs=4096
count=1
1+0 records in
1+0 records out
4096 bytes (4.1 kB, 4.0 KiB) copied, 0.000128036 s, 32.0 MB/s
root@secguest1:/etc# chmod u=rx,go-rwx /etc/luks
root@secguest1:/etc# chmod u=r,go-rwx /etc/luks/boot_os.keyfile
```

Figure 8-10 New LUKS key file

2. By using **cryptsetup luksDump**, inspect the partition that is holding the encrypted data (in this case, /dev/vda6). Figure 8-11 shows that only a single key slot is used. LUKs supply multiple keys, each in a different slot. You must have the passphrase that was used at installation for the disk encryption.

```
# cryptsetup luksDump /dev/vda6
LUKS header information
Version:      2
Epoch:       3
Metadata area: 16384 [bytes]
Keyslots area: 16744448 [bytes]
UUID:        f3eb33ab-7503-41d4-ae3-4e286a1e881d
Label:       (no label)
Subsystem:   (no subsystem)
Flags:       (no flags)

Data segments:
 0: crypt
   offset: 16777216 [bytes]
   length: (whole device)
   cipher: aes-xts-plain64
   sector: 512 [bytes]

Keyslots:
0: luks2
  Key:      512 bits
  Priority:  normal
  Cipher:   aes-xts-plain64
  Cipher key: 512 bits
  PBKDF:    argon2i
  Time cost: 4
  Memory:   354693
  Threads:  2
  Salt:     9b 41 28 6f 4b a5 a8 ac 42 eb f0 37 4e ba 18 06
           b2 d5 a7 62 15 8d d3 0f ad b5 06 0e a1 96 1f 37
  AF stripes: 4000
  AF hash:   sha256
  Area offset: 32768 [bytes]
  Area length: 258048 [bytes]
  Digest ID: 0
Tokens:
Digests:
----- Output truncated for readability -----
```

Figure 8-11 The `cryptsetup luksDump` with single key slot used

3. Add the new key file to an empty LUKS key slot. Figure 8-12 shows this process that uses the **cryptsetup luksAddKey** command. Again, you need the passphrase that was used at installation and is used at start to decrypt the root file system.

```
# cryptsetup luksAddKey /dev/vda6 /etc/luks/boot_os.keyfile
Enter any existing passphrase:
#
```

Figure 8-12 cryptsetup luksAddKey

4. After the new key is added, run **cryptsetup luksDump** again. Now, two key slots are used (see Figure 8-13 on page 266).

```

# cryptsetup luksDump /dev/vda6
LUKS header information
Version:      2
Epoch:       4
Metadata area: 16384 [bytes]
Keyslots area: 16744448 [bytes]
UUID:        f3eb33ab-7503-41d4-ae3-4e286a1e881d
Label:       (no label)
Subsystem:   (no subsystem)
Flags:       (no flags)
Data segments:
  0: crypt
    offset: 16777216 [bytes]
    length: (whole device)
    cipher: aes-xts-plain64
    sector: 512 [bytes]
Keyslots:
  0: luks2
    Key:      512 bits
    Priority: normal
    Cipher:   aes-xts-plain64
    Cipher key: 512 bits
    PBKDF:    argon2i
    Time cost: 4
    Memory:   354693
    Threads:  2
    Salt:     9b 41 28 6f 4b a5 a8 ac 42 eb f0 37 4e ba 18 06
              b2 d5 a7 62 15 8d d3 0f ad b5 06 0e a1 96 1f 37
    AF stripes: 4000
    AF hash:   sha256
    Area offset: 32768 [bytes]
    Area length: 258048 [bytes]
    Digest ID: 0
  1: luks2
    Key:      512 bits
    Priority: normal
    Cipher:   aes-xts-plain64
    Cipher key: 512 bits
    PBKDF:    argon2i
    Time cost: 4
    Memory:   585601
    Threads:  2
    Salt:     0c 5b f4 d3 20 b8 5e e9 9a f4 63 62 5a 0c ad d7
              48 41 85 53 97 1d c2 6d 48 39 1d 43 94 bd 93 b3
    AF stripes: 4000
    AF hash:   sha256
    Area offset: 290816 [bytes]
    Area length: 258048 [bytes]
    Digest ID: 0
Tokens:
Digests:
---- Output truncated for readability ----

```

Figure 8-13 Two key slots in use

5. Add the `KEYFILE_PATTERN` to the `/etc/cryptsetup-initramfs/conf-hook` and set the permissions (see Figure 8-14). Notice that the pattern does not call out a single file; instead, all files that end in `.keyfile` are called out.

```
# echo "KEYFILE_PATTERN=/etc/luks/*.keyfile" >>
/etc/cryptsetup-initramfs/conf-hook
# echo "UMASK=0077" >> /etc/initramfs-tools/initramfs.conf
#
```

Figure 8-14 Setup `initramfs` hook

6. The `/etc/crypttab` is updated to include the new key file. The before and after contents are shown in Figure 8-15 for this modification. Check your `/etc/crypttab` because a mistake might leave your system in a condition in which it cannot be started.

```
root@secquest1:/etc# cat /etc/crypttab
vda6_crypt UUID=f3eb33ab-7503-41d4-ae3-4e286a1e881d none luks,discard
root@secquest1:/etc# vi /etc/crypttab
root@secquest1:/etc# cat /etc/crypttab
vda6_crypt UUID=f3eb33ab-7503-41d4-ae3-4e286a1e881d /etc/luks/boot_os.keyfile
luks,discard
```

Figure 8-15 Update `/etc/crypttab`

7. Update the `initrd` (`initramfs`), as shown in Figure 8-16. Notice how `zipl` is automatically run at the end of this process. Whenever the `initrd` is rebuilt, `zipl` must be run to pick up the newly built `initrd`.

```
# update-initramfs -u
update-initramfs: Generating /boot/initrd.img-5.4.0-42-generic
Using config file '/etc/zipl.conf'
Building bootmap in '/boot'
Building menu 'menu'
Adding #1: IPL section 'ubuntu' (default)
Adding #2: IPL section 'old'
Preparing boot device: vda (0000).
Done.
#
```

Figure 8-16 Update `initramfs`

Now, you should be able to restart your server and *not* be prompted for a passphrase. If you are prompted, review these steps.

It is important *not* to stop here. While the root file system is encrypted, the key to decrypt it is stored in the `initrd` in the clear. When you use the `genprotimg` command, you encrypt this `initrd` so that the key cannot be stolen.

8.7.3 Editing the domain.xml to include iommu='on'

Every virtio device in the domain XML definition must have `iommu='on'` added to it. Also, `memballoon` and Crypto AP pass-through are unsupported in a Secure Execution environment. Before making any changes, we backed up our current domain `.xml`, as shown in Figure 8-17.

```
# virsh dumpxml secguest1 > secguest1-original.xml
#
```

Figure 8-17 Back up domain .xml

Next, by using `virsh edit`, the domain `.xml` was customized to have `iommu='on'` for every virtio device. The memory balloon was disabled, and the QEMU guest agent definition removed. Because we did not plan to use the random number generator, that generator also was removed.

The modified version of the domain `.xml` devices section is shown in Figure 8-18.

```
<devices>
  <emulator>/usr/bin/qemu-system-s390x</emulator>
  <disk type='file' device='disk'>
    <driver name='qemu' type='qcow2' cache='none' io='native' iommu='on' />
    <source file='/var/lib/libvirt/images/secguest1.qcow2' index='1' />
    <backingStore />
    <target dev='vda' bus='virtio' />
    <alias name='virtio-disk0' />
    <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0000' />
  </disk>
  <controller type='pci' index='0' model='pci-root'>
    <alias name='pci.0' />
  </controller>
  <interface type='network'>
    <mac address='52:54:00:3e:65:4c' />
    <source network='default' portid='51695453-c4a4-444a-9195-a603f26dc0cb'
bridge='virbr0' />
    <target dev='vnet0' />
    <model type='virtio' />
    <driver name='qemu' iommu='on' />
    <alias name='net0' />
    <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0001' />
  </interface>
  <console type='pty' tty='/dev/pts/0'>
    <source path='/dev/pts/0' />
    <target type='sclp' port='0' />
    <alias name='console0' />
  </console>
  <memballoon model='none' />
  <panic model='s390' />
</devices>
```

Figure 8-18 Tailored domain xml example

8.7.4 Obtaining the host key documents for the CEC

The following host key documents are needed for this process:

- ▶ The host key document that you received from your cloud provider, HKD-<mmm-nnn>.cert or downloaded from IBM Resource Link. (See: obtaining a host key document from Resource Link).
- ▶ The DigiCert CA certificate, DigiCertCA.crt
- ▶ The IBM Z signing key, ibm-z-host-key.crt
- ▶ The certificate revocation list (CRL), ibm-z-host-key.crl

You can download the CA certificate, the signing key, and the CRL from this [IBM Resource Link web page](#).

Use the sample script that is available [from s390-tools](#) to perform the verification steps.

8.7.5 Validating the key material

It is important to validate the key material for your host and guest. For more information, see this [IBM Documentation web page](#).

There is also a script available to aid with this process at this [GitHub web page](#).

Begin with the `openssl verify` operations, as shown in Figure 8-19. The `openssl verify` is performed on the Certificate Authority and signing key certificate. It also shows extracting the public key from the certificate in the last command.

```
root@secquest1:~/secure_execution# ls
DigiCertCA.crt HKD-8561-022B7F8.crt ibm-z-host-key.crl
ibm-z-host-key-signing.crt
root@secquest1:~/secure_execution# openssl verify -crl_download -crl_check
DigiCertCA.crt
DigiCertCA.crt: OK
root@secquest1:~/secure_execution# openssl verify -crl_download -crl_check
-untrusted DigiCertCA.crt ibm-z-host-key-signing.crt
ibm-z-host-key-signing.crt: OK
root@secquest1:~/secure_execution#
```

Figure 8-19 `openssl verify`

Figure 8-20 on page 270 shows the following steps:

- ▶ Extracting the public signing key into a .pem file
- ▶ Extracting the host key signature from the host key document
- ▶ The use of the resulting value <n> to extract the host key signature into a file that is called signature
- ▶ Extracting the host key document body into a file that is called body
- ▶ Verifying the signature by using the signature and body files

```

root@secguest1:~/secure_execution# openssl x509 -in ibm-z-host-key-signing.crt
-bpubkey -noout > pubkey.pem

root@secguest1:~/secure_execution# openssl asn1parse -in HKD-8561-022B7F8.crt
| tail -1 | cut -d : -f 1
837

root@secguest1:~/secure_execution# openssl asn1parse -in HKD-8561-022B7F8.crt
-out signature -strparse 837 -noout

root@secguest1:~/secure_execution# openssl asn1parse -in HKD-8561-022B7F8.crt
-out body -strparse 4 -noout
root@secguest1:~/secure_execution# openssl sha512 -verify pubkey.pem -signature
signature body
Verified OK

```

Figure 8-20 Verifying the signature of the host key document

The next step is to verify the signature of the host key document issuer. Figure 8-21 shows these steps. The expected output of the last command is shown in the following example:

```

subject= /C=US/ST=New York/L=Poughkeepsie/O=International Business Machines
Corporation/OU=IBM Z Host Key Signing Service/CN=International Business Machines
Corporation

```

Any other value is considered suspect.

```

root@secguest1:~/secure_execution# openssl x509 -in HKD-8561-022B7F8.crt
--issuer -noout
issuer=C = US, O = International Business Machines Corporation, OU = IBM Z Host
Key Signing Service, L = Poughkeepsie, ST = New York, CN = International
Business Machines Corporation
root@secguest1:~/secure_execution# openssl x509 -in ibm-z-host-key-signing.crt
-subject -noout
subject=C = US, ST = New York, L = Poughkeepsie, O = International Business
Machines Corporation, OU = IBM Z Host Key Signing Service, CN = International
Business Machines Corporation

```

Figure 8-21 Verifying host key document issuer

Validating the dates of the signing certificate is shown in Figure 8-22.

```

root@secguest1:~/secure_execution# openssl x509 -in ibm-z-host-key-signing.crt
-dates -noout
notBefore=Apr 17 00:00:00 2020 GMT
notAfter=Apr 22 12:00:00 2022 GMT
root@secguest1:~/secure_execution#

```

Figure 8-22 Validating dates of the signing certificate

Validating the CRL is shown in Figure 8-23.

```
root@secgquest1:~/secure_execution# openssl crl -in ibm-z-host-key.crl -issuer
-noout
issuer=C = US, O = International Business Machines Corporation, OU = IBM Z Host
Key Signing Service, L = Poughkeepsie, ST = New York, CN = International
Business Machines Corporation
root@secgquest1:~/secure_execution# openssl crl -in ibm-z-host-key.crl
-lastupdate -nextupdate -noout
lastUpdate=May 12 14:08:42 2020 GMT
nextUpdate=Jun 15 04:00:00 2020 GMT
root@secgquest1:~/secure_execution#
root@secgquest1:~/secure_execution# openssl crl -in ibm-z-host-key.crl -text
-noout | grep "Serial Number"
    Serial Number: 23AC1FB677F8932BF8
root@secgquest1:~/secure_execution# openssl x509 -in HKD-8561-022B7F8.crt
-serial -noout
serial=16525232812EFD90F5
root@secgquest1:~/secure_execution#
```

Figure 8-23 Validate the CRL

8.7.6 Building a secured initrd image file by using genproting on KVM guest

This secured initrd image file is encrypted by using the key material that can be decrypted only by the specific machine with the corresponding keys. This process is done by the IBM Z or IBM LinuxONE by using the host public key during the start process.

To generate a protected image file, we must add to the command-line parameter. Figure 8-24 shows the original kernel parameter line.

```
root@secgquest1:~# cat /proc/cmdline
root=/dev/mapper/vgsecgquest1-root crashkernel=196M
root@secgquest1:~#
```

Figure 8-24 Kernel parameters

In Figure 8-25, the required swiotbl keyword is added to the text file that is called parmfile. This parmfile is added to the genproting program.

```
root@secgquest1:~# cat parmfile
root=/dev/mapper/vgsecgquest1-root crashkernel=196M swiotlb=262144
```

Figure 8-25 Adding swiotlb keyword

Next, we run the `genprotimg` command. The `-i` parameter is the Linux kernel image that you want to supply as input. The optional `-r` parameter points to the initial RAM disk to be used as input.

The required `-k` parameter supplies the input host key documents. You must supply at least one host-key document. The `-o` parameter is the output file name for your protected virtualization image. An example is shown in Figure 8-26.

```
# genprotimg -i /boot/vmlinuz -r /boot/initrd.img -p parmfile -k
./secure_execution/HKD-8561-022B7F8.crt -o /boot/secure-linux --no-verify -V

WARNING: host-key document verification is disabled. Your workload is not
secured.
   kernel:0x000000015000 (      8093696 /      8090168 Bytes)
   parmline:0x0000007cd000 (       4096 /          66 Bytes)
   ramdisk:0x0000007ce000 (    23719936 /    23715997 Bytes)
   stage3b:0x000001e6d000 (       8192 /       5498 Bytes)
   stage3a:0x000000010000 (       20480 /       20480 Bytes)
#
```

Figure 8-26 `genprotimg` execution

Whenever you regenerate the protect image file `/boot/secure-linux` or any `initrd` file, you must run the `zipl` command to update the boot information. Because we are making `zipl` boot menu changes in the next section, we defer that process for now.

8.7.7 Updating guest `zipl` to boot with secured image in IBM Secure Execution mode

Because we are new to IBM Secure Execution, we decided to initially build a `zipl` menu that defaults to IBM Secure Execution (but can also be overridden) to start without IBM Secure Execution. It is important to remove this configuration file later on when we are sure we can start successfully.

Leaving start options that start an unprotected image makes you vulnerable. For example, your encryption or decryption key might be stolen. Figure 8-27 on page 273 shows the original guest `zipl.conf`.


```
root@secguest1:/etc# cat zipl.conf
[defaultboot]
defaultmenu = menu

:menu
target = /boot
1 = ubuntu
2 = old
default = 1
prompt = 1
timeout = 10

[ubuntu]
target = /boot
image = /boot/vmlinuz
ramdisk = /boot/initrd.img
parameters = root=/dev/mapper/vgsecguest1-root crashkernel=196M

[old]
target = /boot
image = /boot/vmlinuz.old
ramdisk = /boot/initrd.img.old
parameters = root=/dev/mapper/vgsecguest1-root crashkernel=196M
optional = 1
root@secguest1:/etc#
```

Figure 8-27 Original guest zipl.conf

Figure 8-28 shows the modified version. In the modified version, you see the new section that is called [secure].

```
root@secguest1:/etc# cat zipl.conf
[defaultboot]
defaultmenu = menu

:menu
target = /boot
1 = secure
2 = ubuntu
3 = old
default = 1
prompt = 1
timeout = 15

[secure]
target = /boot
image = /boot/secure-linux

[ubuntu]
target = /boot
image = /boot/vmlinuz
ramdisk = /boot/initrd.img
parameters = root=/dev/mapper/vgsecgust1-root crashkernel=196M

[old]
target = /boot
image = /boot/vmlinuz.old
ramdisk = /boot/initrd.img.old
parameters = root=/dev/mapper/vgsecgust1-root crashkernel=196M
optional = 1
root@secguest1:/etc#
```

Figure 8-28 *zipl.conf* with default to IBM Secure Execution

Now that the start menu is modified, the **zipl** command must be run to write out the new **zipl** start information. Figure 8-29 on page 275 shows running **zipl** with the **-V** (verbose) parameter.

```

root@secguest1:/etc# zipl -V
Using config file '/etc/zipl.conf'
Target device information
Device.....: fc:00
Partition.....: fc:05
Device name.....: vda
Device driver name.....: virtblk
Type.....: disk partition
Disk layout.....: SCSI disk layout
Geometry - start.....: 1052672
File system block size.....: 4096
Physical block size.....: 512
Device size in physical blocks..: 1497088
Building bootmap in '/boot'
Building menu 'menu'
Adding #1: IPL section 'secure' (default)
kernel image.....: /boot/secure-linux
component address:
heap area.....: 0x00002000-0x00005fff
stack area.....: 0x0000f000-0x0000ffff
internal loader.: 0x0000a000-0x0000dfff
parameters.....: 0x00009000-0x000091ff
kernel image....: 0x00010000-0x01e7efff
Adding #2: IPL section 'ubuntu'
initial ramdisk...: /boot/initrd.img
kernel image.....: /boot/vmlinuz
kernel parmline...: 'root=/dev/mapper/vgsecguest1-root crashkernel=196M'
component address:
heap area.....: 0x00002000-0x00005fff
stack area.....: 0x0000f000-0x0000ffff
internal loader.: 0x0000a000-0x0000dfff
parameters.....: 0x00009000-0x000091ff
kernel image....: 0x00010000-0x007c6fff
parmline.....: 0x007c8000-0x007c81ff
initial ramdisk.: 0x007d0000-0x01e6e1ff
Adding #3: IPL section 'old'
initial ramdisk...: /boot/initrd.img.old
kernel image.....: /boot/vmlinuz.old
kernel parmline...: 'root=/dev/mapper/vgsecguest1-root crashkernel=196M'
component address:
heap area.....: 0x00002000-0x00005fff
stack area.....: 0x0000f000-0x0000ffff
internal loader.: 0x0000a000-0x0000dfff
parameters.....: 0x00009000-0x000091ff
kernel image....: 0x00010000-0x007c6fff
parmline.....: 0x007c8000-0x007c81ff
initial ramdisk.: 0x007d0000-0x01e6e1ff
Preparing boot device: vda (0000).
Detected SCSI PCBIOS disk layout.
Writing SCSI master boot record.
Syncing disks...
Done.

```

Figure 8-29 Running zipl -V

Now, you should be fully shutting down your guest and restarting it in Secure Execution mode. At this stage, we are performing an initial test. If you fail to start, remember you can select start option #2 to start in normal mode.

If you are successful, do *not* stop here. Some remaining steps must be completed to further secure the configuration because we left the start loader menu entries in place that start without IBM Secure Execution and are unencrypted.

8.7.8 Removing any boot option that is not in Secure Execution mode

After Secure Execution mode is successfully running, it is important to always to remove any option that might not be in Secure Execution.

In Example 8-6, you can see that only the single boot entry exists for our protected image, which is called `secure-linux`.

Example 8-6 Single boot entry for protected image

```
root@secquest1:/etc# cat zipl.conf
[defaultboot]
defaultmenu = menu

:menu
target = /boot
l = secure
default = 1
prompt = 0
timeout = 1

[secure]
target = /boot
image = /boot/secure-linux
```

Note: It is important to run the `zipl` command because the `zipl` configuration file was changed.

8.7.9 Further Ubuntu guest hardening

To further harden this guest, we disabled the emergency and rescue shells. In Example 8-7, you can see that we masked the `systemd` services for emergency and rescue.

Example 8-7 Masking systemd services

```
root@secquest1:/etc# systemctl mask emergency.service
Created symlink /etc/systemd/system/emergency.service · /dev/null.
root@secquest1:/etc# systemctl mask emergency.target
Created symlink /etc/systemd/system/emergency.target · /dev/null.
root@secquest1:/etc# systemctl mask rescue.service
Created symlink /etc/systemd/system/rescue.service · /dev/null.
root@secquest1:/etc# systemctl mask rescue.target
Created symlink /etc/systemd/system/rescue.target · /dev/null.
root@secquest1:/etc#
```

We added `loglevel=0` and `systemd.show_status=no` to the `parmfile`, which is input to `genprotimg`. The modified `parmfile` is shown in Example 8-8.

Example 8-8 Modified parmfile

```
root@sequest1:~# cat parmfile
root=/dev/mapper/vgsequest1-root crashkernel=196M swiotlb=262144 loglevel=0
systemd.show_status=no
```

Only secure remote login can be enforced. SSHD and the SSH keys can be set up and login on kernel consoles can be disabled by disabling serial and virtual TTYs. However, your Linux virtual server cannot be reached if TCP/IP becomes unavailable.

Incorporate the following `systemd` changes to update the `parmfile`:

```
# cat /etc/systemd/system/serial-getty@.service.d/disable.conf
```

```
[Unit]
```

```
ConditionKernelCommandLine=allowlocallogin
```

```
# cat /etc/systemd/system/autovt@.service.d/disable.conf
```

```
[Unit]
```

```
ConditionKernelCommandLine=allowlocallogin
```

Because the `parmfile` was updated, the `secure-linux` image must be rebuilt. The process that is shown in Example 8-9 is the same as is described in 8.7.6, “Building a secured `initrd` image file by using `genprotimg` on KVM guest” on page 271.

Example 8-9 Rebuilding secure-linux image

```
genprotimg -i /boot/vmlinuz -r /boot/initrd.img-5.4.0-42-generic -p
/root/parmfile -k /root/secure_execution/HKD-8561-022B7F8.crt -o
/boot/secure-linux --no-verify -V
WARNING: host-key document verification is disabled. Your workload is not secured.
  kernel:0x00000015000 ( 8093696 / 8090168 Bytes)
  parmline:0x0000007cd000 ( 4096 / 100 Bytes)
  ramdisk:0x0000007ce000 ( 23719936 / 23716354 Bytes)
  stage3b:0x000001e6d000 ( 8192 / 5498 Bytes)
  stage3a:0x00000010000 ( 20480 / 20480 Bytes)
```

Note: Because the `secure-linux` image is now rebuilt, you must run the `zip1` program again.

8.7.10 Removing older unencrypted artifacts from the /boot partition

Because the /boot partition is unencrypted, it is important to remove files that a malicious individual can use to attempt to start the virtual server without the use of IBM Secure Execution.

Before removing files from /boot, it suggested to place a copy somewhere that is secure. For this purpose, we created a directory that is called /root/boot and copied the entire contents of /boot in to this new directory. The file system that is in /root is fully encrypted. In this way, the necessary input files are available if you want to regenerate secure-linux.

Next, we use the `srm` program to securely erase files in /boot that are no longer needed. These files include the `initrd` and `vm linux` files and related symbolic links.

In Example 8-10, you can see the final contents of the /boot file system. The only file in /boot that contains the encryption and decryption key is `secure-linux`, which is encrypted. The others are “securely” deleted with the `srm` program.

Example 8-10 Final /boot file system

```
root@secguest1:/boot# ls -la
total 34284
drwxr-xr-x  3 root root    4096 Sep  8 15:56 .
drwxr-xr-x 19 root root    4096 Aug  3 13:33 ..
-rw-----  1 root root   43520 Sep  8 15:53 bootmap
-rw-r--r--  1 root root   90430 Jul  9 19:50 config-5.4.0-42-generic
drwx-----  2 root root   16384 Aug  3 13:27 lost+found
-rw-r--r--  1 root root 31911936 Sep  8 15:51 secure-linux
-rw-----  1 root root 3088846  Jul  9 19:50 System.map-5.4.0-42-generic
root@secguest1:/boot#
```

8.8 Enabling a SLES 15 SP2 KVM Guest for IBM Secure Execution

All of the tasks that are described in this section are to be performed by the KVM Guest virtual server administrator. The one exception is the domain XML for the guest must be edited to include `iommu='on'` on each virtio device. This task is performed by the KVM Host administrator or cloud provider.

Note: We do not use the Pervasive Encryption approach with protected key technology because Crypto AP pass-through is not supported by IBM Secure Execution at the time of this writing.

Be aware that more steps are needed to be performed with SUSE SLES than with other Linux distributions. These extra steps are needed is because the process uses `grub2` and `zip1` bootloaders. Because they both use the same image and are built with the same process, we must secure both of these bootloaders.

In this example, we use the following process:

- ▶ Install a standard Linux guest on encrypted disk storage.
- ▶ Update KVM guest `/etc/crypttab` to avoid entering password at start.
- ▶ Edit the `domain.xml` to include `iommu='on'`.
- ▶ Obtain the host key documents for the CEC.
- ▶ Validate the key material.
- ▶ Build a secured `initrd` image file by using `genprotimg` on KVM guest.
- ▶ Update the guest `zip1` to start with a secured image in Secure Execution mode.
- ▶ Update the guest `grub2` configuration to start with the secure image.
- ▶ Remove any start option that is *not* in Secure Execution mode.
- ▶ Remove older unencrypted artifacts from `/boot`.

The steps in this process are described next.

8.8.1 Installing a standard Linux guest on encrypted disk storage

We used `virt-install` to deploy a guest in a single command. This process created an 8 Gb qcow2 image file (see Figure 8-30).

```
virt-install --name secquest2q --memory 4096 --disk size=8 --cdrom
/var/lib/libvirt/images/SLE-15-SP2-Full-s390x-GM-Media1.iso
WARNING CDRom media does not print to the text console by default, so you
likely will not see text install output. You might want to use --location. See
the man page for examples of using --location with CDRom media

Starting install...
Allocating 'secquest2q.qcow2'
| 8.0 GB 00:00:00
Connected to domain secquest2q
Escape character is ^]
[ 0.090584] Linux version 5.3.18-22-default (geeko@buildhost) (gcc version
7.5.0 (SUSE Linux)) #1 SMP Wed Jun 3 12:16:43 UTC 2020 (720aeba)
[ 0.090586] setup: Linux is running under KVM in 64-bit mode
[ 0.090596] setup: The maximum memory size is 4096MB
[ 0.090614] numa: NUMA mode: plain
[ 0.090656] cpu: 1 configured CPUs, 0 standby CPUs
[ 0.090717] Write protected kernel read-only data: 10300k
[ 0.090758] Zone ranges:
```

Figure 8-30 One line installation command

During the installation process, we needed to specially tailor the installation. In Figure 8-31 on page 281, you see that we started with the current proposal and custom tailored the partitioning. We needed to make the /partition encrypted.

Note: The installation figures that are used in this section show a graphical installation. Therefore, the VNC in the guest must be configured. For more information about installing VNC, see this SUSE Documentation [web page](#).

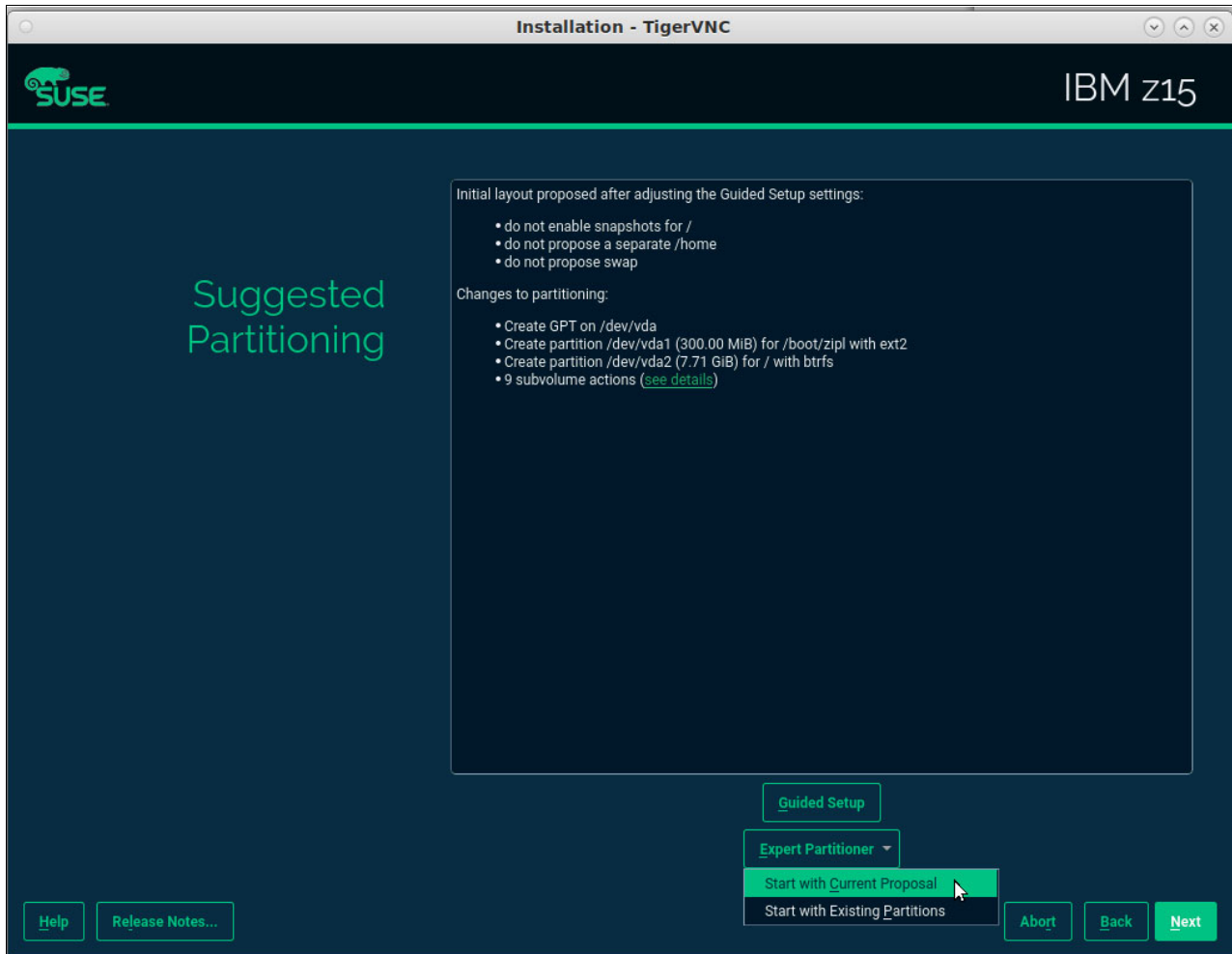


Figure 8-31 Starting with the current proposal

In Figure 8-32 on page 282, you can see we selected the **Encrypt Device** option. Although you might want to make other customizations, the partition was not changed in any other way.

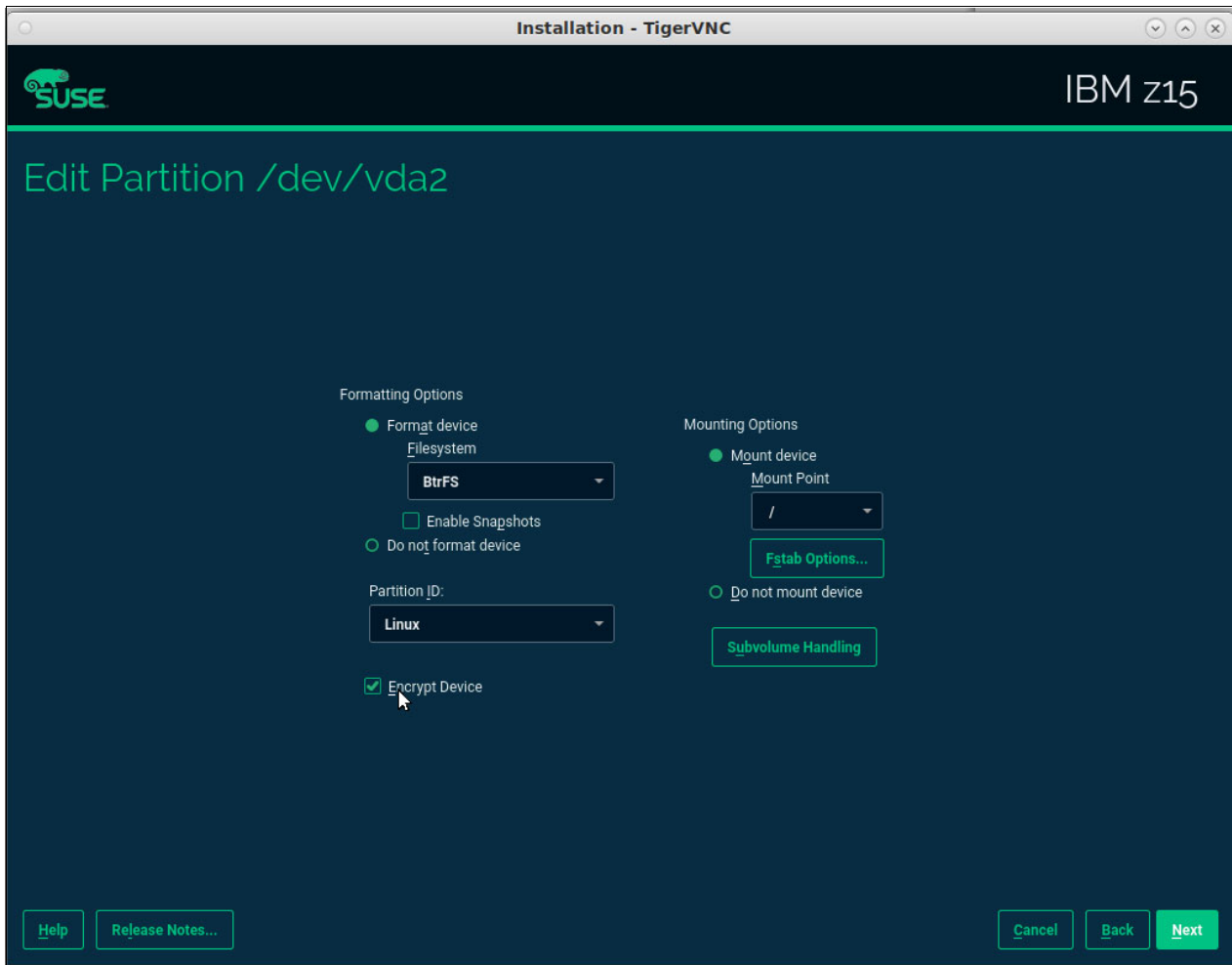


Figure 8-32 Encrypt device selected

You must supply a passphrase to decrypt the /partition, as shown in Figure 8-33. You need this passphrase later to initially start the server. Because the server is to be updated later, you do *not* need to manually supply this password during start.

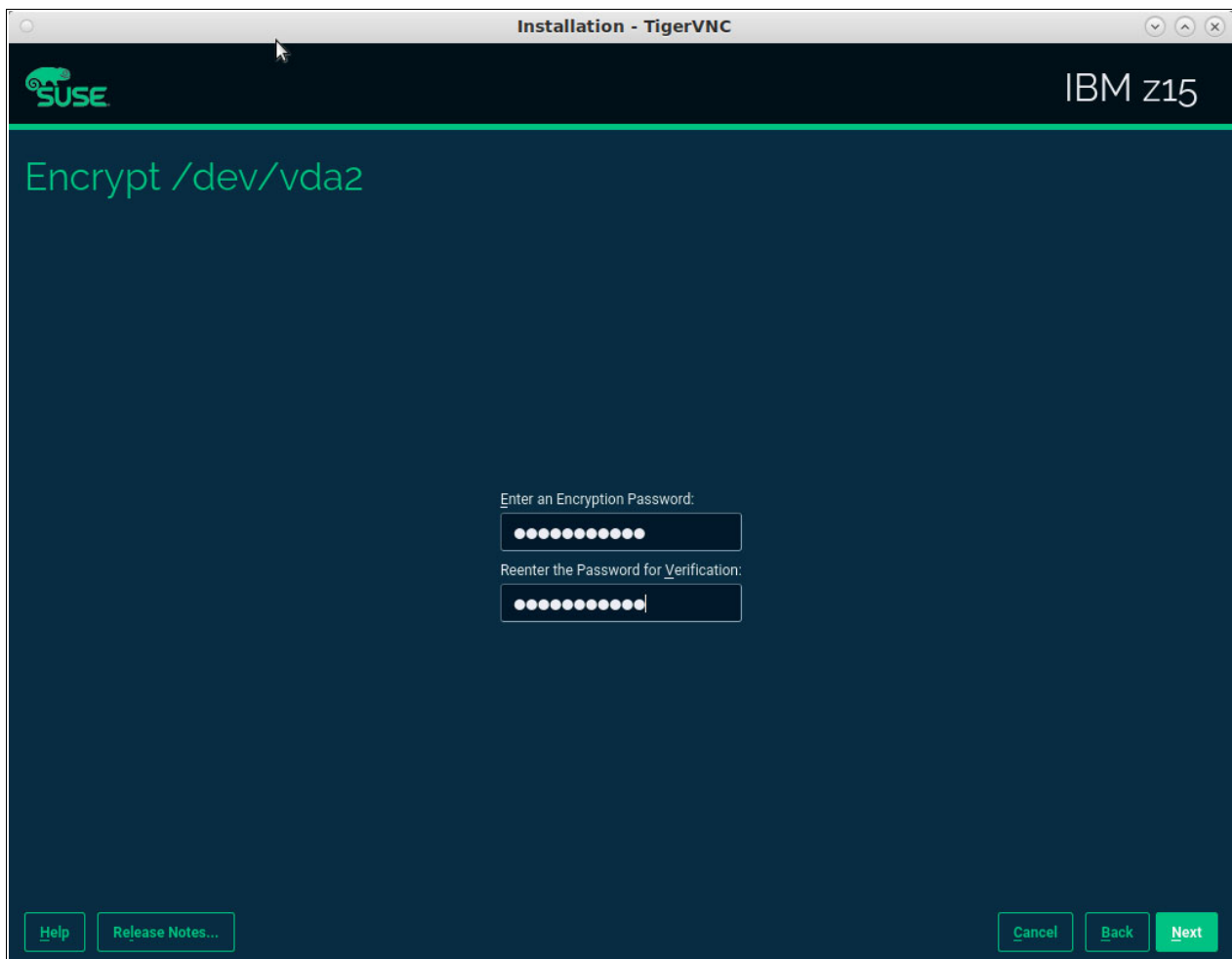


Figure 8-33 Supplying the passphrase

The partition layout for /dev/vda is shown on Figure 8-34. In addition to the /partition, the other partition is /boot/zipl. This partition remains unencrypted, but contains only the zipl boot loader artifacts. The grub2 boot loader artifacts are in /boot/ and /boot/grub2. Both of these are in the encrypted /filesystem. Also, the zipl boot loader phase is responsible for decrypting the /filesystem.

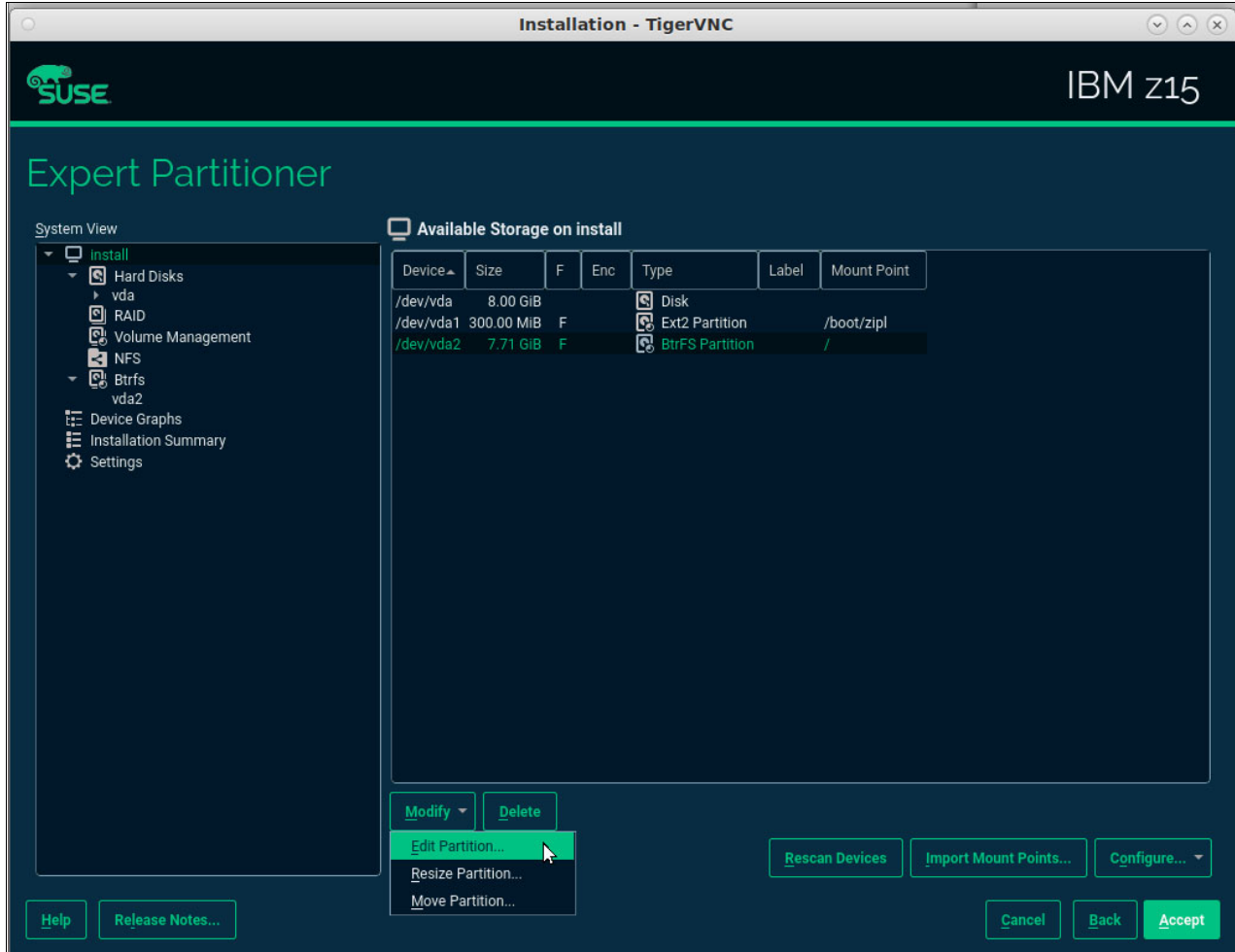


Figure 8-34 Partition layout

It is important to understand that the initial ramdisk (initrd) that is used by grub2 and zipl is the same. The SLES 15 installation and maintenance processes update both of these, which include LUKs keys to the boot disk. Therefore, both must be secured with `genprotimg`.

You might also note no swap partition or device are defined. You can optionally add swap later, but this swap must be encrypted to help keep you virtual server secured. You can encrypt swap disks with random ephemeral Protected Keys. (No CryptoExpress Adapter is required for these keys because they are ephemeral.)

After the installation completes, the server restarts and you must supply the passphrase when prompted to decrypt the root file system. The need to enter the passphrase is temporary because the process is to be automated later.

In Example 8-11, you can see we are starting a server and supplying the passphrase.

Example 8-11 Passphrase prompt to decrypt

```
rdbkkvms:~ # virsh start secquest2q --console
Domain secquest2q started
Connected to domain secquest2q
Escape character is ^]
[ 1.276394] hypfs: The hardware system does not support hypfs
[ 1.276405] hypfs: Initialization of hypfs failed with rc=-61
Please enter passphrase for disk cr_root!
```

After supplying the passphrase, you are presented with the grub2 boot loader menu. After you proceed past this menu and the start is continuing, you see a second prompt for the passphrase to decrypt the root file system (see Example 8-12).

Example 8-12 Second passphrase prompt

```
[ OK ] Started udev Coldplug all Devices.
      Starting Show Plymouth Boot Screen...
      Starting dracut initqueue hook...
[ 1.738250] virtio_blk virtio2: [vda] 16777216 512-byte logical blocks (8.59
GB/8.00 GiB)
[ 1.748446] vda: vda1 vda2
[ 1.767375] scsi host0: Virtio SCSI HBA
[ 1.767711] scsi 0:0:0:0: CD-ROM          QEMU      QEMU CD-ROM      2.5+ PQ:
0 ANSI: 5
[ OK ] Found device /dev/disk/by-path/ccw-0.0.0000-part2.
      Starting Cryptography Setup for cr_root...
[ 1.783038] scsi 0:0:0:0: Attached scsi generic sg0 type 5
[ 1.784359] sr 0:0:0:0: Power-on or device reset occurred
[ 1.784547] sr 0:0:0:0: [sr0] scsi3-mmc drive: 16x/50x cd/rw xa/form2 cdda tray
[ 1.784549] cdrom: Uniform CD-ROM driver Revision: 3.20

Please enter passphrase for disk cr_root!:[ 1.978996] alg: No test for crc32be
(crc32be-vx)
[ 2.024672] EXT4-fs (vda1): mounting ext2 file system using the ext4 subsystem
[ 2.026416] EXT4-fs (vda1): mounted filesystem without journal. Opts: (null)
```

This latest prompt might not be the last message that you receive because of the parallel nature of the Linux start process. However, the start process stops until the correct passphrase is supplied. (The process of manually supplying the decryption phase or key also is planned to be automated.)

Although you now have a virtual server with the root partition encrypted, the `/boot/zipl` is *not* encrypted (see 8.8.11, “Removing unencrypted older artifacts from `/boot/zipl` and encrypted artifacts from `/boot`” on page 302).

Although the root partition is encrypted, we are not operating in a Secure Execution mode yet.

At this point, we shut down the virtual server and made a copy of the qcow2 file as a backup in case any unrecoverable mistakes were made. After the qcow is copied, we restarted the server. Our backup is shown in Figure 8-35.

```
rdbkvm:~/var/lib/libvirt/images # cp secquest2q.qcow2 secquest2q.qcow2.backup
```

Figure 8-35 Backup of new virtual Server

8.8.2 Updating KVM guest /etc/crypttab to avoid entering a password at start

From a security and operational perspective, we do not want any manual prompts at start to which a user must respond. To address this issue, we created a LUKS key file and updated /etc/crypttab to use it. Although initially they both are in the /filesystem, you must rebuild the initrd/initramfs to include them.

With a standard initrd/initramfs, the LUKS key file and updated /etc/crypttab are stored in the clear and present a secure risk. With IBM Secure Execution, this information is encrypted and can be decrypted only in a valid IBM Secure Execution environment.

Figure 8-36 shows the root file system (/), which is derived from cr_root and /dev/vda2.

```
# lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
vda         253:0    0   8G  0 disk
├─vda1      253:1    0 300M  0 part  /boot/zip1
├─vda2      253:2    0 7.7G  0 part
└─cr_root   254:0    0 7.7G  0 crypt /
```

Figure 8-36 Linux Block devices

By using `cryptsetup status`, we can see that cr_root is a LUKS device with aes-xts encryption (see Figure 8-37).

```
# cryptsetup -v status cr_root
/dev/mapper/cr_root is active and is in use.
type:          LUKS1
cipher:        aes-xts-plain64
keysize:       256 bits
key location:  dm-crypt
device:        /dev/vda2
sector size:   512
offset:        4096 sectors
size:          16156639 sectors
mode:          read/write
Command successful.
```

Figure 8-37 Status of cryptsetup

The next step is to create a key file that we can use in `/etc/crypttab`. In Figure 8-38, see that the `/etc/luks` directory and a key file are created, and permissions are set on the directory and file; therefore, only root has read access to this file.

```
secquest2q:/ # mkdir /etc/luks
secquest2q:/ # dd if=/dev/urandom of=/etc/luks/boot_os.keyfile bs=4096 count=1
1+0 records in
1+0 records out
4096 bytes (4.1 kB, 4.0 KiB) copied, 4.8087e-05 s, 85.2 MB/s
secquest2q:/ # chmod u=rx,go-rwx /etc/luks
secquest2q:/ # chmod u=r,go-rwx /etc/luks/boot_os.keyfile
```

Figure 8-38 New LUKS key file

Next, by using `cryptsetup luksDump`, inspect the partition that holds the encrypted data (in this case, `/dev/vda2`). Figure 8-39 shows that only a single key slot is used. LUKS allows users to supply multiple keys, each in a different slot. You the pass phrase that was used at installation for the disk encryption.

```
secquest2q:/ # cryptsetup luksDump /dev/vda2
LUKS header information for /dev/vda2

Version:          1
Cipher name:      aes
Cipher mode:      xts-plain64
Hash spec:        sha256
Payload offset:4096
MK bits:          256
MK digest:        29 49 1f f6 48 b9 36 f8 b8 42 07 ff e7 e6 6c f7 8a d9 a0 a7
MK salt:          7d ea ea 97 4c 6f c7 23 b2 c5 b6 3a ec d1 e3 92
                  ee 34 f8 4b 8d 2c b5 c2 ec 8f 16 5a 54 d9 3a 17
MK iterations:    156038
UUID:             9325f775-6172-4f72-a56f-1002b0646349

Key Slot 0: ENABLED
  Iterations:      2473056
  Salt:            cb b6 36 25 fc da f1 a2 ef 88 88 06 d1 5e 4d 72
                  22 b8 73 8f 34 8e 8c 57 64 59 69 a3 16 30 d7 66
  Key material offset:8
  AF stripes:      4000
Key Slot 1: DISABLED
Key Slot 2: DISABLED
Key Slot 3: DISABLED
Key Slot 4: DISABLED
Key Slot 5: DISABLED
Key Slot 6: DISABLED
Key Slot 7: DISABLED
secquest2q:/ #
```

Figure 8-39 `cryptsetup luksDump` with single key slot in use

The next step is to add our new key file to an empty LUKS key slot. Figure 8-40 shows this addition with the **cryptsetup luksAddKey** command. Again, you must supply the pass phrase that is used at installation and is being used at start to decrypt the root file system.

```
secquest2q:/ # cryptsetup luksAddKey /dev/vda2 /etc/luks/boot_os.keyfile
Enter any existing passphrase:
secquest2q:/ #
```

Figure 8-40 *cryptsetup luksAddKey*

After the new key is added, you can run **cryptsetup luksDump** again. Two key slots are now used.

```
secquest2q:/ # cryptsetup luksDump /dev/vda2
LUKS header information for /dev/vda2

Version:          1
Cipher name:      aes
Cipher mode:      xts-plain64
Hash spec:        sha256
Payload offset:4096
MK bits:          256
MK digest:        29 49 1f f6 48 b9 36 f8 b8 42 07 ff e7 e6 6c f7 8a d9 a0 a7
MK salt:          7d ea ea 97 4c 6f c7 23 b2 c5 b6 3a ec d1 e3 92
                  ee 34 f8 4b 8d 2c b5 c2 ec 8f 16 5a 54 d9 3a 17
MK iterations:    156038
UUID:             9325f775-6172-4f72-a56f-1002b0646349

Key Slot 0: ENABLED
  Iterations:      2473056
  Salt:            cb b6 36 25 fc da f1 a2 ef 88 88 06 d1 5e 4d 72
                  22 b8 73 8f 34 8e 8c 57 64 59 69 a3 16 30 d7 66
  Key material offset:8
  AF stripes:      4000
Key Slot 1: ENABLED
  Iterations:      2467236
  Salt:            85 6a 94 4f 3d 19 1d 2e 5a e1 8f e3 f0 91 11 21
                  6a aa 8f 5c bf 0c 6d c5 fb fb c8 f7 78 86 63 fd
  Key material offset:264
  AF stripes:      4000
Key Slot 2: DISABLED
Key Slot 3: DISABLED
Key Slot 4: DISABLED
Key Slot 5: DISABLED
Key Slot 6: DISABLED
Key Slot 7: DISABLED
secquest2q:/ #
```

Figure 8-41 *Two key slots in use*

Next, we need to have dracut include our new LUKS key file in the `initrd` when a new `initrd` is generated. We accomplish this task by adding a new file that is called `99-root-key.conf` to the `/etc/dracut.conf.d` directory. This file contains the instructions to include our new LUKS key file (see Figure 8-42).

```
secquest2q:/etc/dracut.conf.d # cat 99-root-key.conf
install_items+=" /etc/luks/boot_os.keyfile "
secquest2q:/etc/dracut.conf.d #
```

Figure 8-42 Setting up dracut to include the key file.

Next, `/etc/crypttab` is updated to include our new key file. Both the before and after contents are shown in Figure 8-43 for this modification.

Note: Check your `/etc/crypttab` because a mistake can leave your system in a condition in which it cannot be restarted.

```
secquest2q:/etc # cat crypttab
cr_root /dev/disk/by-path/ccw-0.0.0000-part2
secquest2q:~/test # cat /etc/crypttab
cr_root /dev/disk/by-path/ccw-0.0.0000-part2 /etc/luks/boot_os.keyfile luks
```

Figure 8-43 Updating `/etc/crypttab`

The next step is to update the `initrd` (see Figure 8-44). Notice how `zipl` is *not* run automatically at the end of this process. Whenever the `initrd` is rebuilt, `zipl` must be run to pick up the newly built `initrd`. Also, note the update location; it does not include `/boot/zipl`.

```
secquest2q:/ # dracut -v -f
dracut: Executing: /usr/bin/dracut -v -f
dracut: dracut module 'dmraid' will not be installed, because command 'dmraid'
could not be found!
dracut: dracut module 'dmsquash-live-ntfs' will not be installed, because
command 'ntfs-3g' could not be found!
dracut: dracut module 'biosdevname' will not be installed, because command
'biosdevname' could not be found!
dracut: dracut module 'dmraid' will not be installed, because command 'dmraid'
could not be found!
dracut: dracut module 'dmsquash-live-ntfs' will not be installed, because
command 'ntfs-3g' could not be found!
dracut: *** Including module: bash ***
dracut: *** Including module: systemd ***
dracut: *** Including module: systemd-initrd ***

<<Lines omitted for readability>>

dracut: lrwxrwxrwx  1 root    root          6 Aug 28 13:29 var/run ->
../run
dracut: drwxr-xr-x  1 root    root          0 Aug 28 13:29 var/tmp
dracut:
=====
dracut: *** Creating initramfs image file '/boot/initrd-5.3.18-22-default' done
***
secquest2q:/ #
```

Figure 8-44 Updating `initrd` with `dracut`

Thus far, the `grub2 boot` loader was addressed and those prompts eliminate; however, the `zipl` boot loader must also be addressed.

To address the `zipl` boot loader, the new `initrd` and image must be placed into `/boot/zipl` and the `zipl` program run to write the updated information. Although you can perform these steps manually, `grub2-install`.

Figure 8-45 shows the `grub2-install` execution. In addition to populating a fresh `initrd` and image in `/boot/zipl`, you can see at the end of the output that it is running the `zipl` command. The `grub2-install` requires the `initrd` and images from `/boot`. If you remove them from `/boot`, `grub2-install` fails.

```
secquest2q:/ # grub2-install
Installing for s390x-emu platform.
dracut: Executing: /usr/bin/dracut --hostonly --force
/boot/zipl/initrd-5.3.18-22-default 5.3.18-22-default
dracut: dracut module 'dmraid' will not be installed, because command 'dmraid'
could not be found!
dracut: dracut module 'dmsquash-live-ntfs' will not be installed, because
command 'ntfs-3g' could not be found!
dracut: dracut module 'biosdevname' will not be installed, because command
'biosdevname' could not be found!
dracut: dracut module 'dmraid' will not be installed, because command 'dmraid'
could not be found!
dracut: dracut module 'dmsquash-live-ntfs' will not be installed, because
command 'ntfs-3g' could not be found!
dracut: *** Including module: bash ***
dracut: *** Including module: systemd ***

<<<text omitted for readability>>>

dracut: *** Generating early-microcode cpio image ***
dracut: *** Store current command line parameters ***
dracut: Stored kernel commandline:
dracut: rd.luks.uuid=luks-9325f775-6172-4f72-a56f-1002b0646349
dracut: rd.zipl=/dev/disk/by-path/ccw-0.0.0000-part1
dracut: root=/dev/mapper/cr_root rootfstype=btrfs
rootflags=rw,relatime,space_cache,subvolid=256,subvol=@,subvol=@
dracut: *** Creating image file '/boot/zipl/initrd-5.3.18-22-default' ***
dracut: *** Creating initramfs image file '/boot/zipl/initrd-5.3.18-22-default'
done ***
Using config file '/boot/zipl/config' (from command line)
Building bootmap in '/boot/zipl'
Building menu 'menu'
Adding #1: IPL section 'grub2' (default)
Adding #2: IPL section 'skip-grub2'
Adding #3: IPL section 'grub2-mem1G'
Adding #4: IPL section 'se'
Preparing boot device: vda (0000).
Done.
Installation finished. No error reported.
secquest2q:/ #
```

Figure 8-45 `grub2-install`

Now, you should be able to restart your server and *not* be prompted for a passphrase at start. If you are prompted, check your steps.

Note: It is important not to stop here. Although the root file system is encrypted, the key to decrypt it is stored in the `initrd` that is in the clear. When you use the `genprotimg` command, you encrypt this `initrd` so the key cannot be stolen.

8.8.3 Editing the domain.xml to include iommu='on'

Every virtio device in the domain XML definition must have `iommu='on'` added to it. Also, `memballoon` and `Crypto AP` pass-through are unsupported in an IBM Secure Execution environment. Before making any changes, we made a backup copy of our current domain `.xml`, as shown in Figure 8-46

```
# virsh dumpxml secguest2q > secguest2q-original.xml
#
```

Figure 8-46 Backup domain xml

Next, by using `virsh edit`, the domain xml was tailored to have `iommu='on'` for every virtio device. The memory balloon was disabled, and the QEMU guest agent definition removed. Also, the random number generator was removed because we did not plan to use it. The modified version of the domain XML devices section is shown in Figure 8-47.

```
<devices>
  <emulator>/usr/bin/qemu-system-s390x</emulator>
  <disk type='file' device='disk'>
    <driver name='qemu' type='qcow2' cache='none' io='native' iommu='on' />
    <source file='/var/lib/libvirt/images/secguest2q.qcow2' index='1' />
    <backingStore />
    <target dev='vda' bus='virtio' />
    <alias name='virtio-disk0' />
    <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0000' />
  </disk>
  <controller type='pci' index='0' model='pci-root'>
    <alias name='pci.0' />
  </controller>
  <interface type='network'>
    <mac address='52:54:00:f5:f7:8c' />
    <source network='default' portid='93bfb1c3-efc1-4b41-b253-b592f98fc3f0'
bridge='virbr0' />
    <target dev='vnet1' />
    <model type='virtio' />
    <driver name='qemu' iommu='on' />
    <alias name='net0' />
    <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0001' />
  </interface>
  <console type='pty' tty='/dev/pts/3'>
    <source path='/dev/pts/3' />
    <target type='sclp' port='0' />
    <alias name='console0' />
  </console>
  <memballoon model='none' />
  <panic model='s390' />
</devices>
```

Figure 8-47 Tailored domain xml example

8.8.4 Obtaining the host key documents for the CEC

The following host key documents are needed:

- ▶ The host key document that you received from your cloud provider, HKD-<mmm-nnn>.crt, or downloaded from Resource Link, see this IBM Resource Link [web page](#).
- ▶ The DigiCert CA certificate, DigiCertCA.crt.
- ▶ The IBM Z signing key, ibm-z-host-key-signing.cr.
- ▶ The certificate revocation list (CRL), ibm-z-host-key.cr.

The CA certificate, the signing key, and the CRL are available from this IBM Resource Link [web page](#).

Use the sample script that is available from [s390-tools](#) to perform the verification steps.

8.8.5 Validating the key material

For more information about the process that is used to validate the key material, see 8.7.5, “Validating the key material” on page 269. This process is the same for all Linux distributions.

8.8.6 Building a secured initrd image file by using genprotimg on KVM guest

This secured `initrd` image file is encrypted by using the key material that can be decrypted only by using the specific machine with the corresponding keys. This process is done by IBM Z or LinuxONE during the start process.

To generate a protected image file, we must make add the statements that are shown in Figure 8-48 to the command line parameter. Figure 8-48 shows the original kernel parameter line.

```
secquest2q:/ # cat /proc/cmdline
root=UUID=637e330e-efd7-4442-8318-701ec6930bd2 TERM=linux console=ttyS0
console=ttyS1 mitigations=auto crashkernel=191M
```

Figure 8-48 Kernel cmdline

In Figure 8-49, the required `swiotlb` keyword is added to the text file that is called `parmfile`. This `parmfile` is incorporated into the `genprotimg` program.

```
secquest2q:/ # cat parmfile
root=UUID=637e330e-efd7-4442-8318-701ec6930bd2 TERM=linux console=ttyS0
console=ttyS1 mitigations=auto crashkernel=191M swiotlb=262144
```

Figure 8-49 Adding `swiotlb` keyword

Next, we run the `genprotimg` command. Consider the following points:

- ▶ The `-i` parameter is the Linux kernel image that you want to supply as input.
- ▶ The optional `-r` parameter point to the initial ram disk to be used as input.

- ▶ The required `-k` parameter supplies the input host key documents. You need to supply at least one host-key document.
- ▶ The `-o` parameter is the output file name for your protected virtualization image. An example is shown in Figure 8-50.

```

secguest2q:~/secure_execution # genprotimg -i /boot/image -r /boot/initrd -p
parmfile -k ./HKD-8561-022B7F8.crt -o /boot/secure-linux --no-verify -V
WARNING: host-key document verification is disabled. Your workload is not
secured.
   kernel:0x000000015000 (   6262784 /   6259256 Bytes)
   parmfile:0x00000060e000 (    4096 /    135 Bytes)
   ramdisk:0x00000060f000 (  9904128 /  9902200 Bytes)
   stage3b:0x000000f81000 (    8192 /    5594 Bytes)
   stage3a:0x000000010000 (    20480 /    20480 Bytes)
secguest2q:~/secure_execution #

```

Figure 8-50 *genprotimg* execution

Whenever you regenerate the protect image file `/boot/secure-linux` or any `initrd` file, you must run the `zipl` command to update the start information. For more information, see [Updating guest zipl to boot with secured image in IBM Secure Execution mode](#).

Note: The `secure-linux` file also *must* be copied from `/boot` to `/boot/zipl` for the `zipl` boot loader to pick it up.

8.8.7 Updating guest zipl to boot with secured image in IBM Secure Execution mode

Because we are new to IBM Secure Execution, we decided to initially build a `zipl` menu that defaults to IBM Secure Execution (but also can be overridden) to boot without IBM Secure Execution. It is important to remove this menu later on when we are sure we can boot successfully.

Leaving boot options that boot an unprotected image makes you vulnerable. For example, your encryption or decryption key can be stolen.

With SUSE SLES 15, the `zipl` configuration file is generated by the system. Figure 8-51 on page 295 shows the beginning of the original `/etc/default/zipl2grub.conf.in` **that is used** to generate the guest `zipl` configuration file.

```
## This is the template for '@zipldir@/config' and is subject to
## rpm's %config file handling in case of grub2-s390x-emu package update.

[defaultboot]
defaultmenu = menu

[grub2]
    target = @zipldir@
    ramdisk = @zipldir@/initrd,0x2000000
    image = @zipldir@/image
    parameters = "root=@GRUB_DEVICE@ @GRUB_EMU_CONMODE@ @GRUB_CMDLINE_LINUX@
@GRUB_CMDLINE_LINUX_DEFAULT@ initgrub quiet splash=silent plymouth.enable=0 "

[grub2-mem1G]
    target = @zipldir@
    image = @zipldir@/image
    ramdisk = @zipldir@/initrd,0x2000000
    parameters = "root=@GRUB_DEVICE@ @GRUB_EMU_CONMODE@ @GRUB_CMDLINE_LINUX@
@GRUB_CMDLINE_LINUX_DEFAULT@ initgrub quiet splash=silent plymouth.enable=0
mem=1G "
```

Figure 8-51 Beginning of original `zipl2grub.conf.in`

We must add an entry to the `thiszipl2grub.conf.in` template to boot with IBM Secure Execution. Figure 8-52 on page 296 shows our modified version of `zipl2grub.conf.in`, which contains a new section that is call `[se]` that points to our `secure-linux` file.

```

## This is the template for '@zipldir@/config' and is subject to
## rpm's %config file handling in case of grub2-s390x-emu package update.

[defaultboot]
defaultmenu = menu

[se]
    target = @zipldir@
    image = @zipldir@/secure-linux
    parameters = "root=@GRUB_DEVICE@ @GRUB_EMU_CONMODE@ @GRUB_CMDLINE_LINUX@
@GRUB_CMDLINE_LINUX_DEFAULT@ initgrub quiet splash=silent plymouth.enable=0 "
[grub2]
    target = @zipldir@
    ramdisk = @zipldir@/initrd,0x2000000
    image = @zipldir@/image
    parameters = "root=@GRUB_DEVICE@ @GRUB_EMU_CONMODE@ @GRUB_CMDLINE_LINUX@
@GRUB_CMDLINE_LINUX_DEFAULT@ initgrub quiet splash=silent plymouth.enable=0 "

[grub2-mem1G]
    target = @zipldir@
    image = @zipldir@/image
    ramdisk = @zipldir@/initrd,0x2000000
    parameters = "root=@GRUB_DEVICE@ @GRUB_EMU_CONMODE@ @GRUB_CMDLINE_LINUX@
@GRUB_CMDLINE_LINUX_DEFAULT@ initgrub quiet splash=silent plymouth.enable=0
mem=1G "

```

Figure 8-52 Beginning of modified `zipl2grub.conf.in`

In addition to modifying the beginning of the `zipl2grub.conf.in`, the end of the file must be updated to include the new `[se]` section in the menu. The end of the original `zipl2grub.conf.in` is shown in Figure 8-53.

```

:menu
    target = @zipldir@
    timeout = 60
    default = 1
    prompt = 0
    secure = @SUSE_SECURE_BOOT@
    1 = grub2
    2 = skip-grub2
    3 = grub2-mem1G
#@    4 = grub2-previous
#@    5 = skip-grub2-previous
#@    6 = grub2-mem1G-previous

```

Figure 8-53 End of original `zipl2grub.conf.in`

The modified version with the new `se` entry is shown in Figure 8-54 on page 297. Notice that we did not change the default entry to boot from (it remains `default=1`).

```
:menu
  target = @zipldir@
  timeout = 60
  default = 1
  prompt = 0
  secure = @SUSE_SECURE_BOOT@
  1 = grub2
  2 = skip-grub2
  3 = grub2-mem1G
  4 = se
#@ 4 = grub2-previous
#@ 5 = skip-grub2-previous
#@ 6 = grub2-mem1G-previous
```

Figure 8-54 End of modified `zipl2grub.conf.in`

Now, we run the `grub2-install` command again. By doing so, the `zipl` configuration file (`/boot/zipl/config`) is rebuilt and the `zipl` command runs against it.

We are *not* ready to start in Secure Execution mode yet. We updated only the `zipl` boot loader and not the `grub2` boot loader, which is described next.

8.8.8 Updating guest `grub2` to boot with secured image in IBM Secure Execution mode

Like the `zipl` boot loader configuration, the `grub2` boot loader also is built from a template. To add our customized entry to this template, update `40_custom` in the `/etc/grub.d` directory. A similar process was used for the SUSE SLES KVM host, but the contents of the updates are different.

Figure 8-55 shows the contents of an unmodified `40_custom` file.

```
!/bin/sh
exec tail -n +3 $0
# This file provides an easy way to add custom menu entries. Simply type the
# menu entries you want to add after this comment. Be careful not to change
# the 'exec tail' line above.
```

Figure 8-55 Original `40_custom grub2` file

To add our new entry, complete the following steps:

1. Copy a section of the `/boot/grub2/grub.cfg` file. After the line that contains: `### BEGIN /etc/grub.d/10_linux ###`, copy the approximate nine lines of information, ending with the closing “}” bracket. (This information is at the bottom of the `40_custom` file.)
2. Update the pasted in content to have a unique menuentry name. Figure 8-56 shows that “SE” was added to the end of the name. The hot key number then is updated to be unique (in our example, a value of 6).

For the `menuentry_id_option`, a long string value is included. In our example, we added “se” to the end.

3. Update the echo “Loading Linux” with a specific kernel level to read “Loading Linux secure-linux”.
4. Update the line that begins with `linux` to point the “`/boot/secure-linux`” file name, and remove the “`initrd`” line.

```
#!/bin/sh
exec tail -n +3 $0
# This file provides an easy way to add custom menu entries. Simply type the
# menu entries you want to add after this comment. Be careful not to change
# the 'exec tail' line above.

menuentry 'SLES 15-SP2 SE' --hotkey=6 --class sles --class gnu-linux --class
gnu --class os $menuentry_id_option
'gnulinux-simple-637e330e-efd7-4442-8318-701ec6930bd2se' {
    set gfxpayload=text
    insmod gzio
    echo 'Loading Linux secure-linux ...'
    linux ${btrfs_subvol}/boot/secure-linux
root=UUID=637e330e-efd7-4442-8318-701ec6930bd2 ${extra_cmdline} TERM=linux co
nsole=ttyS0 console=ttyS1 mitigations=auto crashkernel=191M
}
```

Figure 8-56 Customized `40_custom` grub2 file

5. Build a new grub2 configuration. Figure 8-57 shows running `grub2-mkconfig`.

```
secquest2q:/etc/grub.d # grub2-mkconfig -o /boot/grub2/grub.cfg
Generating grub configuration file ...
Found linux image: /boot/image-5.3.18-22-default
Found initrd image: /boot/initrd-5.3.18-22-default
done
secquest2q:/etc/grub.d #
```

Figure 8-57 `grub2-mkconfig` in guest

6. Run `grub2-once --enum` to determine the boot menu entry for IBM Secure Execution. In Figure 8-58 on page 299, you can see this entry is grub2 menu entry number 2.

```

secguest2q:/etc/grub.d # grub2-once --enum

0          SLES 15-SP2

1>0       Advanced options for SLES 15-SP2>SLES 15-SP2, with Linux
5.3.18-22-default

1>1       Advanced options for SLES 15-SP2>SLES 15-SP2, with Linux
5.3.18-22-default (recovery mode)

2          SLES 15-SP2 SE
secguest2q:/etc/grub.d

```

Figure 8-58 *grub2-once*

As described in 8.8.7, “Updating guest zip1 to boot with secured image in IBM Secure Execution mode” on page 294, you added the zip1 menu entry number 4 for IBM Secure Execution; for grub2, the menu entry number is 2. (We did not change the default settings from which menu entry to boot.)

To select both of these entry numbers at boot, specify loadparm 4g2. From the KVM host, use **virsh edit** to update the domain XML and specify loadparm 4g2, as shown in Figure 8-59. (Loadparm is part of the boot XML tag.)

To use this entry on a disk, you must remove any boot specification from the operating system section of the domain XML.

```

<disk type='file' device='disk'>
  <driver name='qemu' type='qcow2' cache='none' io='native' iommu='on' />
  <source file='/var/lib/libvirt/images/secguest2q.qcow2' />
  <target dev='vda' bus='virtio' />
  <boot order='1' loadparm='4g2' />
  <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0000' />
</disk>

```

Figure 8-59 *loadparm in domain xml*

Now, fully shut down your guest and restart it in Secure Execution mode. At this stage, we are performing an initial test. If the start fails, remove the loadparm specification and start the previous configuration mode.

Some other steps must be completed to further secure the configuration because we left boot loader menu entries in place that start without Secure Execution and are unencrypted.

Before making any further changes, you might want to make another backup copy of your qcow2 image file. Several updates were made, which should be done while the guest is shut down. Figure 8-60 shows a new backup being made.

```

rdbkvm:/var/lib/libvirt/images # cp secguest2q.qcow2 secguest2q.qcow2.backup2

```

Figure 8-60 *qcow backup*

8.8.9 Removing any start option that is not Secure Execution mode

After the guest starts successfully in Secure Execution mode, it is important to always remove any option that is not in Secure Execution. For SUSE SLES, this requirement applies to the `zipl` and `grub2` boot loaders.

Start by updating the `zipl` configuration. In Example 8-13, `/etc/default/zipl2grub.conf.in` was updated so that the `image` line for every menu entry points to `secure-linux`. (The example shows only the update portion, not the entire file.)

Example 8-13 Updated `zipl2grub.conf.in`

```
[defaultboot]
defaultmenu = menu

[se]
  target = @zipldir@
  image = @zipldir@/secure-linux
  parameters = "root=@GRUB_DEVICE@ @GRUB_EMU_CONMODE@ @GRUB_CMDLINE_LINUX@
@GRUB_CMDLINE_LINUX_DEFAULT@ initgrub quiet splash=silent plymouth.enable=0 "
[grub2]
  target = @zipldir@
  image = @zipldir@/secure-linux
  parameters = "root=@GRUB_DEVICE@ @GRUB_EMU_CONMODE@ @GRUB_CMDLINE_LINUX@
@GRUB_CMDLINE_LINUX_DEFAULT@ initgrub quiet splash=silent plymouth.enable=0 "

[grub2-mem1G]
  target = @zipldir@
  image = @zipldir@/secure-linux
  parameters = "root=@GRUB_DEVICE@ @GRUB_EMU_CONMODE@ @GRUB_CMDLINE_LINUX@
@GRUB_CMDLINE_LINUX_DEFAULT@ initgrub quiet splash=silent plymouth.enable=0 mem=1G
"

[skip-grub2]
  target = @zipldir@
  image = @zipldir@/secure-linux
  parameters = "root=@GRUB_DEVICE@ @GRUB_CONMODE@ @GRUB_CMDLINE_LINUX@
@GRUB_CMDLINE_LINUX_DEFAULT@ "
#@
```

Also, be sure to update the entries in `zipl2grub.conf.in` that look like comments. They feature Linux that contains `image = @zipldir@/image.prev`, which you also must update to point to `secure-linux`. Also, remove the `ramdisk` lines in each entry.

After making the updates that are shown in Example 8-13, re-run `grub2-install` for the changes to take effect. After that process is complete, update the `grub2` boot loader.

Modifying the process that automatically builds `/boot/grub2/grub.cfg` takes some planning. The `grub2` programs, such as `grub2-mkconfig`, expect to find `initrd` and `image` files in `/boot`. They also have `initialmenu` entries that are not intended to be customized. The process allows you to add only customized entries.

By editing one line in `/etc/grub.d/10_linux` as shown in Example 8-14, `grub2-mkconfig` builds all new menu entries with `secure-linux`.

Example 8-14 Customized 10_linux

```
case "$machine" in
  xi?86 | xx86_64) klist="/boot/vmlinuz-* /vmlinuz-* /boot/kernel-*" ;;
  xaarch64) klist="/boot/Image-* /Image-* /boot/kernel-*" ;;
  xarm*) klist="/boot/zImage-* /zImage-* /boot/kernel-*" ;;
  xs390 | xs390x) klist="/boot/secure-linux* /boot/secure-linux*" ;;
  *) klist="/boot/vmlinuz-* /boot/vmlinuz-* /vmlinuz-* /vmlinuz-* \
    /boot/kernel-*" ;;
esac
```

After updating `10_linux`, run `grub2-mkconfig`, as shown in Example 8-15. It should recognize only the `secure-linux` image.

Example 8-15 grub2-mkconfig

```
grub2-mkconfig -o /boot/grub2/grub.cfg
Generating grub configuration file ...
Found linux image: /boot/secure-linux
done
```

Now, no menu entries point to any image or `initrd` other than `secure-linux`. This statement is true for `zipl` and `grub2`.

8.8.10 Further SLES 15 guest hardening

To further harden this guest, we disabled the emergency and rescue shells. In Example 8-16, you can see that we masked the `systemd` services for emergency and rescue.

Example 8-16 Mask emergency and rescue

```
secquest2q:/ # systemctl mask emergency.service
Created symlink /etc/systemd/system/emergency.service · /dev/null.
secquest2q:/ # systemctl mask emergency.target
Created symlink /etc/systemd/system/emergency.target · /dev/null.
secquest2q:/ # systemctl mask rescue.service
Created symlink /etc/systemd/system/rescue.service · /dev/null.
secquest2q:/ # systemctl mask rescue.target
Created symlink /etc/systemd/system/rescue.target · /dev/null.
secquest2q:/ #
```

We also added `loglevel=0` and `systemd.show_status=no` to the `parmfile`, which is input to `genprotimg`. The modified `parmfile` is shown in Example 8-17.

Example 8-17 Customized parmfile

```
secquest2q:~/secure_execution # cat parmfile
root=UUID=637e330e-efd7-4442-8318-701ec6930bd2 TERM=linux console=ttyS0
console=ttyS1 mitigations=auto crashkernel=191M swiotlb=262144 loglevel=0
systemd.show_status=no
```

You can enforce secure remote login only. You can set up SSHD and the SSH keys and disable login on kernel consoles by disabling serial and virtual TTYs. However, you cannot reach your Linux virtual server if TCP/IP becomes unavailable.

Incorporate the following systemd changes to update the parmfile:

```
# cat /etc/systemd/system/serial-getty@.service.d/disable.conf
```

```
[Unit]
```

```
ConditionKernelCommandLine=allowlocallogin
```

```
# cat /etc/systemd/system/autovt@.service.d/disable.conf
```

```
[Unit]
```

```
ConditionKernelCommandLine=allowlocallogin
```

Because the parmfile was updated, the secure-linux image must be rebuilt. The process that is shown in Example 8-18 is the same as you performed as described in 8.7.6, “Building a secured `initrd` image file by using `genprotimg` on KVM guest” on page 271.

Example 8-18 Final genprotimg

```
secgquest2q:~/secure_execution # genprotimg -i /boot/image -r /boot/initrd -p
parmfile -k ./HKD-8561-022B7F8.crt -o /boot/secure-linux --no-verify -V
WARNING: host-key document verification is disabled. Your workload is not secured.
  kernel:0x000000015000 ( 6262784 / 6259256 Bytes)
  parmfile:0x000000060e000 ( 4096 / 169 Bytes)
  ramdisk:0x000000060f000 ( 9904128 / 9902200 Bytes)
  stage3b:0x0000000f81000 ( 8192 / 5594 Bytes)
  stage3a:0x000000010000 ( 20480 / 20480 Bytes)
secgquest2q:~/secure_execution #
```

You must manually copy the secure-linux file from `/boot` to `/boot/zipl`, as shown in Example 8-19.

Example 8-19 Copying secure-linux file

```
secgquest2q:~/secure_execution # cp /boot/secure-linux /boot/zipl/secure-linux
secgquest2q:~/secure_execution #
```

Because the secure-linux image is now rebuilt, you must run the `zipl` program again by running `grub2-install`.

8.8.11 Removing unencrypted older artifacts from `/boot/zipl` and encrypted artifacts from `/boot`

The `/boot/zipl` partition is unencrypted. It is important to remove files that a malicious individual might use to try and start the virtual server without Secure Execution.

Before removing files from `/boot`, it is suggested to place a copy in a secure place. To do this, we created a directory that is called `/root/boot` and copied the entire contents of `/boot` into it (the file system in which `/root` is stored is fully encrypted). If you want to regenerate secure-linux, you have the necessary input files.

Next, we use the program **shred** to securely erase files in `/boot/zipl` that are no longer needed. These files include the `initrd` and `image` files and related symbolic links.

The original contents of the `/boot/zipl` file system is shown in Example 8-20.

Example 8-20 Final /boot/zipl filesystem

```
secquest2q:/boot/zipl # ls -la
total 32366
drwxr-xr-x 3 root root    1024 Sep 10 17:13 .
drwxr-xr-x 1 root root     512 Sep 10 16:39 ..
-rw-r--r-- 1 root root      0 Aug 27 19:58 active_devices.txt
-rw----- 1 root root  306176 Sep 10 17:13 bootmap
-rw-r--r-- 1 root root    2114 Sep 10 17:13 config
lrwxrwxrwx 1 root root      23 Sep 10 17:13 image -> image-5.3.18-22-default
-rw-r--r-- 1 root root  6259256 Sep 10 10:33 image-5.3.18-22-default
lrwxrwxrwx 1 root root      12 Sep 10 16:36 image.prev -> secure-linux
lrwxrwxrwx 1 root root      24 Sep 10 17:13 initrd -> initrd-5.3.18-22-default
-rw----- 1 root root 10221580 Sep 10 17:13 initrd-5.3.18-22-default
lrwxrwxrwx 1 root root      24 Sep 10 10:34 initrd.prev ->
initrd-5.3.18-22-default
drwx----- 2 root root   12288 Aug 27 19:54 lost+found
-rw-r--r-- 1 root root 16265216 Sep 10 17:12 secure-linux
secquest2q:/boot/zipl #
```

In Example 8-21, you can see the shredding and removal of files and symbolic links, and the final contents of `/boot/zipl`.

The only remaining file in `/boot/zipl` that contains the encryption and decryption key is `secure-linux`, which is encrypted. The other files were securely deleted by using the `shred` program.

Example 8-21 Shred and final /boot/zip

```
secquest2q:/boot/zipl # shred -u initrd-5.3.18-22-default
secquest2q:/boot/zipl # shred -u image-5.3.18-22-default
secquest2q:/boot/zipl # rm image
secquest2q:/boot/zipl # rm initrd
secquest2q:/boot/zipl # rm initrd.prev
secquest2q:/boot/zipl # ls -la
total 16205
drwxr-xr-x 3 root root    1024 Sep 10 17:22 .
drwxr-xr-x 1 root root     512 Sep 10 16:39 ..
-rw-r--r-- 1 root root      0 Aug 27 19:58 active_devices.txt
-rw----- 1 root root  306176 Sep 10 17:13 bootmap
-rw-r--r-- 1 root root    2114 Sep 10 17:13 config
lrwxrwxrwx 1 root root      12 Sep 10 16:36 image.prev -> secure-linux
drwx----- 2 root root   12288 Aug 27 19:54 lost+found
-rw-r--r-- 1 root root 16265216 Sep 10 17:12 secure-linux
secquest2q:/boot/zipl #
```

Although the contents of `/boot` is encrypted, it is possible to interact with grub and attempt a start from AN `initrd` or `image` and `initrd/image` without IBM Secure Execution.

As an added safety measure, run `shred -u` against the `initrd` and `image` files that are in `/boot` (remembering that you stored a backup copy in a safe place).

Example 8-22 shows the extra steps that were taken.

Example 8-22 Running a shred command against initrd

```
secguest2q:/boot # shred -u image-5.3.18-22-default
secguest2q:/boot # shred -u initrd-5.3.18-22-default
secguest2q:/boot # shred -u initrd-5.3.18-22-default-kdump
secguest2q:/boot # rm image
secguest2q:/boot # rm initrd
```

8.9 Enabling a RHEL KVM Guest for Secure Execution

The tasks in this section are performed by the KVM Guest virtual server administrator with perhaps the exception of one task. The domain XML for the guest must be edited to include `iommu='on'` on each virtio device. This task is performed by the KVM Host administrator or Cloud provider.

Note: We do *not* use the Pervasive Encryption approach with protected key technology because Crypto AP pass-through is not supported by IBM Secure Execution at the time of this writing.

In this example, we perform the following tasks, which are described next:

- ▶ Install a standard Linux guest on encrypted disk storage
- ▶ Update KVM guest `/etc/crypttab` to avoid entering password at boot
- ▶ Edit the `domain.xml` to include `iommu='on'`
- ▶ Obtain the host key documents for the CEC
- ▶ Validate the key material
- ▶ Build a secured `initrd` image file by using `genprotimg` on the KVM guest
- ▶ Update guest `zipl` to start with secured image in Secure Execution mode
- ▶ Remove securely the original unprotected kernel, `initrd`, and `parmfile` files
- ▶ Further harden RHEL guest
- ▶ Remove unencrypted legacy artifacts from `/boot`

8.9.1 Installing a standard Linux guest on encrypted disk storage

Complete the following steps:

1. Create your qcow2 file as disk for the secured guest (see Example 8-61).

```
[root@rdbkvm1 isos]# qemu-img create -f qcow2 kvm1secguest01_vol001.img 10G
Formatting 'kvm1secguest01_vol001.img', fmt=qcow2 size=10737418240
cluster_size=65536 lazy_refcounts=off refcount_bits=16
```

Figure 8-61 Creating the qcow2 file

2. Use `virt-install` to deploy a guest in a single command (see Example 8-62).

```
root@rdbkvm1 isos]# virt-install --name kvm1secguest02 --memory 4000 --vcpus 2
--os-variant rhel8.4 --import --disk path=/home/isos/kvm1secguest01_vol001.img
--network network:macvtap-net1 --location
/var/ftp/pub/RHEL-8.4.0-20210503.1-s390x-dvd1.iso --boot
kernel=/mnt/rhel84/images/kernel.img,initrd=/mnt/rhel84/images/initrd.img
--extra-args "ip=129.40.23.201::129.40.23.254:255.255.255.0::enc1:none"
```

Figure 8-62 One line installation command

Note: For a VNC graphical installation, you must pass extra kernel parameters (ip= statement). Therefore, you must specify the kernel and initrd location (they are in the image directory of the .iso file)

To simplify the disk installation, we chose the guided selection for disk storage that is shown in Example 8-63. You must select the Encrypt my data option.

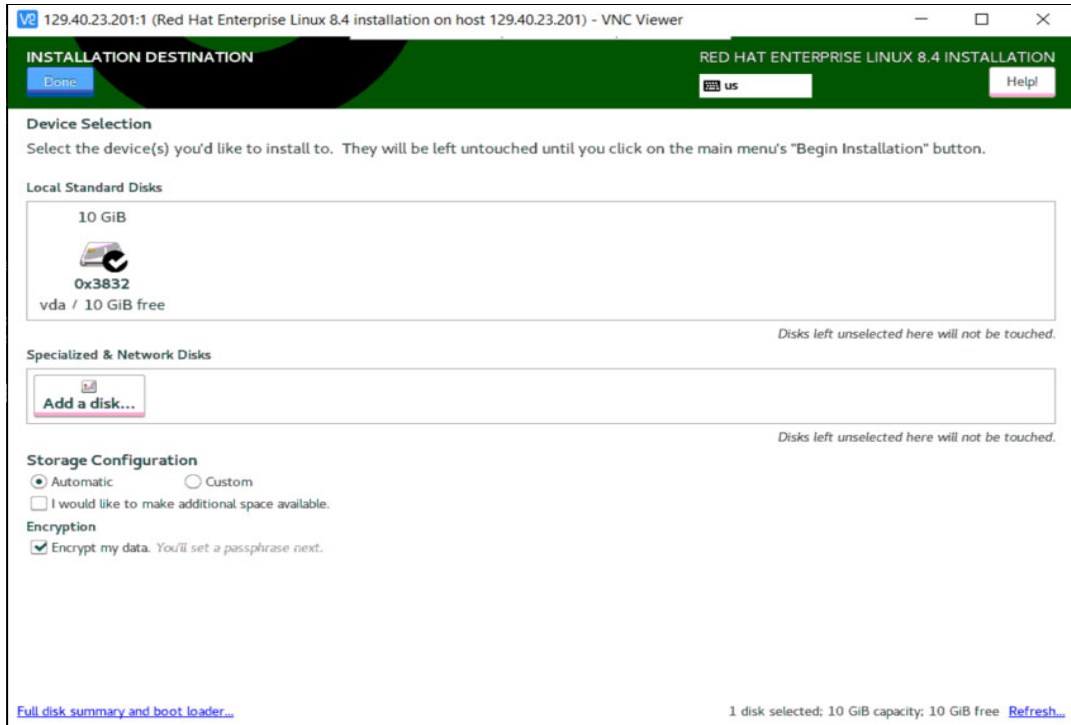


Figure 8-63 Selecting Encrypt my data option

The installation process through the VNC is shown in Example 8-64, including the encryption of the /dev/vda2 for the root file system.

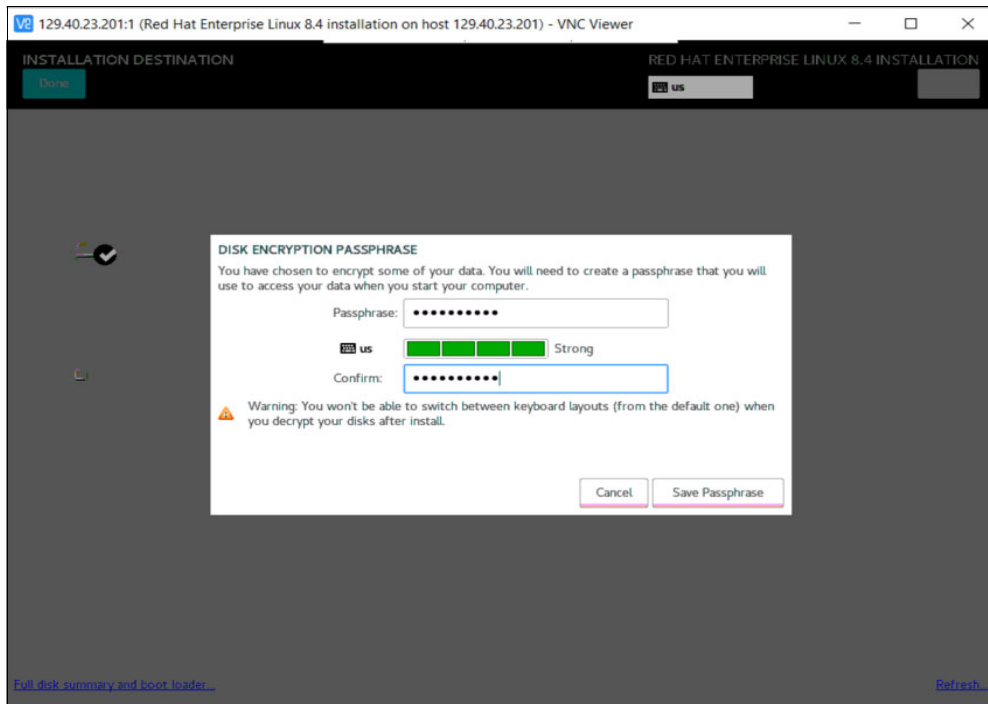


Figure 8-64 /dev/vda2 encrypted for the root file system

We provided an Encryption Phrase. You must remember to store this phrase in a safe place. With this phrase, your root file system can be decrypted (1nx4rdbk21).

After the installation finishes, you must remove the kernel and initrd parameters from the guest definition by using **virsh edit** to start normally, as shown in Figure 8-65.

```
<os>
  <type arch='s390x' ma-chine='s390-ccw-virtio-rhel8.2.0'>hvm</type>
  <kernel>/mnt/rhel84/images/kernel.img</kernel>
  <initrd>/mnt/rhel84/images/initrd.img</initrd>
  <boot dev='hd' />
</os>
```

Figure 8-65 Removing the kernel and the initrd statements from the domain definition

3. After the installation completes, the server restarts and you must supply the pass phrase when prompted to decrypt the root file system, as shown in Example 8-66. However, this process is temporary because it is to be automated later.

```
[root@rdbkvm1 images]# virsh start kvm1secguest02 --console
Domain kvm1secguest02 started
Connected to domain kvm1secguest02
Escape character is ^]

Please enter passphrase for disk luks-8898ca51-2780-4c6e-a959-75ad7ecb4eac!:
```

Figure 8-66 Passphrase prompt to decrypt disk

You now have a virtual server with the root partition encrypted (/boot is *not* encrypted) Although. Although the root partition in encrypted, we are not operating in a Secure Execution mode yet.

4. Shut down the virtual server and make a copy of the qcow2 file as a backup in case any unrecoverable mistakes were made. After the qcow2 file is copied, restart the server as a back up, as shown in Figure 8-67.

```
[root@rdbkvm1 isos]# cp kvm1secguest01_vol001.img
kvm1secguest01_vol001.img.bkp
```

Figure 8-67 Executing a backup of new virtual server

8.9.2 Updating KVM guest /etc/crypttab to avoid entering password at boot

From a security and an operational perspective, we do not want any manual prompts at start to which a user must respond. To address this issue, we created a LUKS key file and updated /etc/crypttab to use it.

Although both of these files are in /filesystem, you must rebuild the initrd/initramfs to include them.

With a standard initrd/initramfs, these are stored in the clear and present a secure risk. With Secure Execution, this information is encrypted and can be decrypted only in a valid IBM Secure Execution environment.

Figure 8-68 shows that the volume group for root and swap are derived from luks-8898ca51-2780-4c6e-a959-75ad7ecb4eac.

```
[root@kvm1secguest02 ~]# lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE
MOUNTPOINT
sr0                                  11:0    1 1024M  0 rom
vda                                  252:0    0   10G  0 disk
??vda1                              252:1    0    1G  0 part  /boot
??vda2                              252:2    0    9G  0 part
  ??luks-8898ca51-2780-4c6e-a959-75ad7ecb4eac 253:0    0    9G  0 crypt
    ??rhel-root                       253:1    0    8G  0 lvm   /
    ??rhel-swap                       253:2    0    1G  0 lvm   [SWAP]
```

Figure 8-68 Linux Block devices

By using **cryptsetup status**, we can see that vda6_crypt is a LUKS2 device with aes-xts encryption, as shown in Figure 8-69.

```
[root@kvm1secguest02 ~]# cryptsetup -v status
luks-8898ca51-2780-4c6e-a959-75ad7ecb4eac
/dev/mapper/luks-8898ca51-2780-4c6e-a959-75ad7ecb4eac is active and is in use.
type:      LUKS2
cipher:    aes-xts-plain64
keysize:   512 bits
key location: keyring
device:    /dev/vda2
sector size: 512
offset:    32768 sectors
size:      18839552 sectors
mode:      read/write
flags:     discards
Command successful.
```

Figure 8-69 cryptsetup status

Next, we create a key file that can be used in `/etc/crypttab`. As shown in Figure 8-70, the `/etc/luks` directory and a key file are created, and permissions are set on the directory and file so that only root has read access to this file.

```
root@kvm1secguest02 ~]# mkdir /etc/luks
[root@kvm1secguest02 ~]# dd if=/dev/urandom of=/etc/luks/boot_os.keyfile
bs=4096 count=1
1+0 records in
1+0 records out
4096 bytes (4.1 kB, 4.0 KiB) copied, 7.8801e-05 s, 52.0 MB/s
root@secguest1:/etc# chmod u=rx,go-rwx /etc/luks
root@secguest1:/etc# chmod u=r,go-rwx /etc/luks/boot_os.keyfile
```

Figure 8-70 New luks key file

Next, by using `cryptsetup luksDump`, inspect the partition that is holding the encrypted data (in our example `/dev/vda2`). Figure 8-71 on page 310 shows that only a single key slot is currently uses. LUKS allows use to supply multiple keys each in a different slot. You need the pass phrase that is used during installation for the disk encryption.

```

[root@kvm1secguest02 ~]# cryptsetup luksDump /dev/vda2
LUKS header information
Version:          2
Epoch:           3
Metadata area:   16384 [bytes]
Keyslots area:  16744448 [bytes]
UUID:            8898ca51-2780-4c6e-a959-75ad7ecb4eac
Label:           (no label)
Subsystem:       (no subsystem)
Flags:           (no flags)

Data segments:
 0: crypt
   offset: 16777216 [bytes]
   length: (whole device)
   cipher: aes-xts-plain64
   sector: 512 [bytes]

Keyslots:
 0: luks2
   Key:          512 bits
   Priority:     normal
   Cipher:       aes-xts-plain64
   Cipher key:  512 bits
   PBKDF:        argon2i
   Time cost:    4
   Memory:       542037
   Threads:      2
   Salt:         1f 1c 8d 03 30 f9 1a 90 50 b8 fe 3c 43 44 fe 9b
                  db c9 11 fb 83 62 06 5a a0 90 43 79 67 3f 43 6e
   AF stripes:  4000
   AF hash:      sha256
   Area offset: 32768 [bytes]
   Area length: 258048 [bytes]
   Digest ID:   0
----- Output truncated for readability -----

```

Figure 8-71 *cryptsetup luksDump* with single key slot in use

The next step is to add our new key file to an empty luks key slot. Figure 8-72 shows this addition with the **cryptsetup luksAddKey** command. Again, you must supply the pass phrase that was used at installation and is used now at start to decrypt the root file system.

```

[root@kvm1secguest02 ~]# cryptsetup luksAddKey /dev/vda2
/etc/luks/boot_os.keyfile
Enter any existing passphrase:

```

Figure 8-72 *cryptsetup luksAddKey*

After the new key is added, you can run **cryptsetup luksDump** again. Now, two key slots are used (see Figure 8-73 on page 311).

```

[root@kvm1secguest02 ~]# cryptsetup luksDump /dev/vda2
LUKS header information
Version:          2
Epoch:           4
Metadata area:   16384 [bytes]
Keyslots area:  16744448 [bytes]
UUID:            8898ca51-2780-4c6e-a959-75ad7ecb4eac
Label:           (no label)
Subsystem:       (no subsystem)
Flags:           (no flags)

Data segments:
 0: crypt
   offset: 16777216 [bytes]
   length: (whole device)
   cipher: aes-xts-plain64
   sector: 512 [bytes]

Keyslots:
 0: luks2
   Key:          512 bits
   Priority:     normal
   Cipher:       aes-xts-plain64
   Cipher key:  512 bits
   PBKDF:        argon2i
   Time cost:   4
   Memory:      542037
   Threads:     2
   Salt:        1f 1c 8d 03 30 f9 1a 90 50 b8 fe 3c 43 44 fe 9b
                db c9 11 fb 83 62 06 5a a0 90 43 79 67 3f 43 6e
   AF stripes:  4000
   AF hash:     sha256
   ..... Area offset:32768 [bytes]
            Area length:258048 [bytes]
            Digest ID: 0
 1: luks2
   Key:          512 bits
   Priority:     normal
   Cipher:       aes-xts-plain64
   Cipher key:  512 bits
   PBKDF:        argon2i
   Time cost:   4
   Memory:      541560
   Threads:     2
   Salt:        24 d6 ae 6f 3d ac aa 01 3d c0 2e db 0d 95 6b 90
                1d e6 52 bb cb 43 95 c1 06 f5 d9 a8 e5 8b 0f 70
   AF stripes:  4000
   AF hash:     sha256
   Area offset:290816 [bytes]
   Area length:258048 [bytes]
   Digest ID: 0
---- Output truncated for readability ----

```

Figure 8-73 Two key slots in use

The next step is to include `/etc/luks/boot_os.keyfile` in the `initrd` to avoid the prompting of the root file system encrypting password. You must add a file in the `/etc/dracut.conf.d` directory to prepare the addition of the key file into the `initrd` (see Figure 8-74).

```
[root@kvm1secguest02 ~]# cat /etc/dracut.conf.d/99-root-key.conf
install_items+=" /etc/luks/boot_os.keyfile"
```

Figure 8-74 Adding configuration file to the dracut

Next, `/etc/crypttab` is updated to include our new key file. The before and after contents are shown in Figure 8-75 for this modification. Recheck your `/etc/crypttab` because a mistake can leave your system in a condition in which it cannot be started.

```
----- BEFORE -----
[root@kvm1secguest02 ~]# cat /etc/crypttab
luks-8898ca51-2780-4c6e-a959-75ad7ecb4eac
UUID=8898ca51-2780-4c6e-a959-75ad7ecb4eac none luks,discard
----- AFTER -----
[root@kvm1secguest02 ~]# vim /etc/crypttab
[root@kvm1secguest02 ~]# cat /etc/crypttab
luks-8898ca51-2780-4c6e-a959-75ad7ecb4eac
UUID=8898ca51-2780-4c6e-a959-75ad7ecb4eac /etc/luks/boot_os.keyfile
luks,discard
```

Figure 8-75 Update `/etc/crypttab`

The next step is to update the `initrd` (that is, `initramfs`) by using the `dracut` command and update the `zipl` (see Figure 8-76).

```
[root@kvm1secguest02 dracut.conf.d]# dracut -f
[root@kvm1secguest02 dracut.conf.d]# zipl -V
```

Figure 8-76 Update `initramfs` and run the `zipl`

Now, you should be able to restart your server and *not* be prompted for a passphrase at start. If you are prompted, recheck your steps.

While the root file system is encrypted, the key to decrypt it is stored in the `initrd` in the clear. When you use the `genprotimg` command, you encrypt this `initrd` so that the key cannot be stolen.

8.9.3 Editing the domain.xml to include `iommu='on'`

Every virtio device in the domain XML definition must have `iommu='on'` added to it. Also, `memballoon` and `Crypto AP` pass-through are unsupported in a Secure Execution environment. Before making any changes, backed up our domain `.xml`, as shown in Figure 8-77.

```
[root@rdbkvm1 isos]# virsh dumpxml kvm1secguest02 >
kvm1secguest02-original.xml
```

Figure 8-77 Copy of current domain xml

Next, by using `virsh edit`, the domain `.xml` was tailored to have `iommu='on'` for every virtio device. The memory balloon was disabled, and the QEMU guest agent definition was removed.

Also, the random number generator was removed because we did not plan to use it. The modified version of the domain XML devices section is shown in Figure 8-78.

```
<devices>
  <emulator>/usr/libexec/qemu-kvm</emulator>
  <disk type='file' device='disk'>
    <driver name='qemu' type='qcow2' iommu='on'/>
    <source file='/home/isos/kvm1secguest01_vol001.img' index='1'/>
    <backingStore/>
    <target dev='vda' bus='virtio'/>
    <alias name='virtio-disk0'/>
    <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0000'/>
  </disk>
  <controller type='pci' index='0' model='pci-root'>
    <alias name='pci.0'/>
  </controller>
  <interface type='direct'>
    <mac address='52:54:00:87:89:9d'/>
    <source network='macvtap-net1'
portid='e31abc8d-6c0f-46f2-b721-95fc90da0bdd' dev='bond1.008' mode='bridge'/>
    <target dev='macvtap36'/>
    <model type='virtio'/>
    <driver name='qemu' iommu='on'/>
    <alias name='net0'/>
    <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0001'/>
  </interface>
  <console type='pty' tty='/dev/pts/1'>
    <source path='/dev/pts/1'/>
    <target type='sc1p' port='0'/>
    <alias name='console0'/>
  </console>
  <memballoon model='none'/>
  <panic model='s390'/>
</devices>
```

Figure 8-78 Tailored domain xml example

8.9.4 Obtaining the host key documents from the CEC

The following four host key documents are need for this task:

- ▶ The host key document that you received from your cloud provider (HKD-`<mmm-nnn>.crt`) or downloaded from the Resource Link.
- ▶ DigiCert CA certificate: `DigicertCA.crt`
- ▶ IBM Z signing key: `ibm-z-host-key-signing.crt`
- ▶ Certificate revocation list (CRL): `ibm-z-host-key.crl`

You can download the CA certificate, signing key, and CRL from this [IBM Resource Link web page](#).

Use the sample script that is available from this [GitHub web page](#) to perform the verification steps.

8.9.5 Validating the key material

It is important to validate the key material for your host and guest. For more information, see this [IBM Documentation web page](#).

A script also is available to aid with this process at this [GitHub web page](#).

Begin with the `openssl verify` operations that are shown in Figure 8-79. The `openssl` verification process is performed on the Certificate Authority and signing key certificate. It also shows extracting the public key from the certificate in the last command.

```
[root@kvm1secguest02 root]# ls
DigiCertCA.crt HKD-8561-022B7F8.crt ibm-z-host-key.crt
ibm-z-host-key-revocation-list ibm-z-host-key-signing.crt
[root@kvm1secguest02 root]# openssl verify -crl_download -crl_check
DigiCertCA.crt
[root@kvm1secguest02 root]# openssl verify -crl_download -crl_check -untrusted
DigiCertCA.crt ibm-z-host-key-signing.crt
ibm-z-host-key-signing.crt: OK
```

Figure 8-79 Opening SSL verification

Figure 8-80 shows the following next steps:

- ▶ Extracting the public signing key into a `.pem` file
- ▶ Extracting the host key signature from the host key document
- ▶ Using the resulting value `<n>` to extract the host key signature into a file called `signature`
- ▶ Extracting the host key document body into a file that is called `body`
- ▶ Verifying open SSL verification
- ▶ The signature by using the `signature` and `body` files

```
[root@kvm1secguest02 root]# openssl x509 -in ibm-z-host-key-signing.crt
-pubkey -noout > pubkey.pem
[root@kvm1secguest02 root]# openssl asn1parse -in HKD-8561-022B7F8.crt | tail
-1 | cut -d : -f 1
837
[root@kvm1secguest02 root]# openssl asn1parse -in HKD-8561-022B7F8.crt -out
signature -strparse 837 -noout
[root@kvm1secguest02 root]# openssl asn1parse -in HKD-8561-022B7F8.crt -out
body -strparse 4 -noout
[root@kvm1secguest02 root]# openssl sha512 -verify pubkey.pem -signature
signature body
Verified OK
```

Figure 8-80 Verifying the signature of the host key document

The next steps are to verify the signature of the host key document issuer. Figure 8-81 shows these steps. The expected output of the last command is:

```
subject= /C=US/ST=New York/L=Poughkeepsie/O=International Business Machines Corporation/OU=IBM Z Host Key Signing Service/CN=International Business Machines Corporation
```

Any other value is considered suspect.

```
[root@kvm1secguest02 root]# openssl x509 -in HKD-8561-022B7F8.crt --issuer -noout
issuer=C = US, O = International Business Machines Corporation, OU = IBM Z Host Key Signing Service, L = Poughkeepsie, ST = New York, CN = International Business Machines Corporation
[root@kvm1secguest02 root]# openssl x509 -in ibm-z-host-key-signing.crt -subject -noout
subject=C = US, ST = New York, L = Poughkeepsie, O = International Business Machines Corporation, OU = IBM Z Host Key Signing Service, CN = International Business Machines Corporation
[root@kvm1secguest02 root]# openssl x509 -in ibm-z-host-key-signing.crt -dates -noout
notBefore=Apr 17 00:00:00 2020 GMT
notAfter=Apr 22 12:00:00 2022 GMT
```

Figure 8-81 Verifying host key document issuer

Any other value is considered suspect (see Figure 8-82).

```
[root@kvm1secguest02 root]# openssl crl -in ibm-z-host-key.crl -issuer -noout
issuer=C = US, O = International Business Machines Corporation, OU = IBM Z Host Key Signing Service, L = Poughkeepsie, ST = New York, CN = International Business Machines Corporation
[root@kvm1secguest02 root]# openssl crl -in ibm-z-host-key.crl -lastupdate -nextupdate -noout
lastUpdate=Nov 13 05:00:16 2021 GMT
nextUpdate=Dec 15 05:00:00 2021 GMT
[root@kvm1secguest02 root]# openssl crl -in ibm-z-host-key.crl -text -noout | grep "Serial Number"
Serial Number: 23AC1FB677F8932BF8
[root@kvm1secguest02 root]# openssl x509 -in HKD-8561-022B7F8.crt -serial -noout
serial=16525232812EFD90F5
```

Figure 8-82 Validating the CRL

8.9.6 Building a secured initrd image file using genproting on KVM guest

This secured `initrd` image file is encrypted by using the key material that can be decrypted only by the specific machine with the corresponding keys. This process is done by IBM Z or LinuxONE during the start process.

To generate a protected image file, we must add to the command line parameter. Figure 8-83 shows the original kernel parameter line.

```
[root@kvm1secguest02 root]# cat /proc/cmdline
root=/dev/mapper/rhel-root crashkernel=auto
rd.luks.uuid=luks-8898ca51-2780-4c6e-a959-75ad7ecb4eac rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap
```

Figure 8-83 Original kernel parameters

In Figure 8-84, the required `swiotlb` keyword is added to the text file called `parmfile`, which is imported into the `genproting` program.

```
[root@kvm1secguest02 root]# vim parmfile
[root@kvm1secguest02 root]# cat parmfile
root=/dev/mapper/rhel-root crashkernel=auto swiotlb=262144
rd.luks.uuid=luks-8898ca51-2780-4c6e-a959-75ad7ecb4eac rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap
```

Figure 8-84 Creating a `parmfile` with `swiotlb` added parameter

Next, we run the `genproting` command, which can include the following parameters:

- ▶ The `-i` parameter is the Linux kernel image that you want to supply as input.
- ▶ The optional `-r` parameter points to the initial ram disk to be used as input.
- ▶ The required `-k` parameter supplies the input host key documents. You must supply at least one host-key document.
- ▶ The `-o` parameter is the output file name for your protected virtualization image. An example is shown in Figure 8-85.

```
[root@kvm1secguest02 root]# genproting -i /boot/vmlinuz-4.18.0-305.el8.s390x -r
/boot/initramfs-4.18.0-305.el8.s390x.img -p /home/root/parmfile -k
/home/root/HKD-8561-022B7F8.crt -o /boot/secure-linux --no-verify -V
WARNING: host-key document verification is disabled. Your workload is not
secured.
   kernel:  0x000000015000 (    7221248 /    7217637 Bytes)
  parmfile: 0x00000006f8000 (     4096 /        154 Bytes)
   ramdisk: 0x00000006f9000 (  28921856 /  28920502 Bytes)
   stage3b: 0x000000228e000 (     8192 /     5386 Bytes)
   stage3a: 0x000000010000 (    20480 /    20480 Bytes)
```

Figure 8-85 `genproting` execution

Whenever you regenerate the protect image file `/boot/secure-linux` or any `initrd` file, you must update the start options that point to the protected image and run the `zip1` to update the start menu. Because we are changing the `zip1` start menu as described next section, we defer that save for now.

8.9.7 Updating guest zipl to boot with secured image in Secure Execution mode

Because we are new to IBM Secure Execution, we decided to initially build a `zipl` menu that defaults to Secure Execution. We update the only start option to be secure so that no unsecured start options exist. Figure 8-86 shows the original boot loader entries.

```
[root@kvm1secguest02 ~]# ls /boot/loader/entries -l
total 8
-rw-r--r--. 1 root root 545 Dec  9 16:58
9a84c2b961c340c492dcf817435f5a74-0-rescue.conf
-rw-r--r--. 1 root root 455 Dec  9 19:21
9a84c2b961c340c492dcf817435f5a74-4.18.0-305.el8.s390x.conf

[root@kvm1secguest02 entries]# cat
9a84c2b961c340c492dcf817435f5a74-4.18.0-305.el8.s390x.conf
title Red Hat Enterprise Linux (4.18.0-305.el8.s390x) 8.4 (Ootpa)
version 4.18.0-305.el8.s390x
linux /boot/vmlinuz-4.18.0-305.el8.s390x
initrd /boot/initramfs-4.18.0-305.el8.s390x.img
options root=/dev/mapper/rhel-root crashkernel=auto swiotlb=262144
rd.luks.uuid=luks-8898ca51-2780-4c6e-a959-75ad7ecb4eac rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap
id rhel-20210429132122-4.18.0-305.el8.s390x
grub_users $grub_users
grub_arg --unrestricted
grub_class kernel
```

Figure 8-86 Checking the latest boot configuration

Figure 8-87 shows the modified version.

```
[root@kvm1secguest02 ~]# cat
/boot/loader/entries/9a84c2b961c340c492dcf817435f5a74-4.18.0-305.el8.s390x.conf
title Red Hat Enterprise Linux (4.18.0-305.el8.s390x) 8.4 (Ootpa)
version 4.18.0-305.el8.s390x
linux /boot/secure-linux
id rhel-20210429132122-4.18.0-305.el8.s390x
grub_users $grub_users
grub_arg --unrestricted
grub_class kernel
```

Figure 8-87 Modified boot entry

Now that the start menu was modified, the `zipl` command is run to write out the new `zipl` start information.

Figure 8-88 shows running `zipl` with the `-V` (verbose) parameter.

```
[root@kvm1secgquest02 entries]# zipl -V
Using config file '/etc/zipl.conf'
Using BLS config file
'/boot/loader/entries/9a84c2b961c340c492dcf817435f5a74-4.18.0-305.e18.s390x.conf'
Using BLS config file
'/boot/loader/entries/9a84c2b961c340c492dcf817435f5a74-0-rescue.conf'
Target device information
Device.....: fc:00
Partition.....: fc:01
Device name.....: vda
Device driver name.....: virtblk
Type.....: disk partition
Disk layout.....: SCSI disk layout
Geometry - start.....: 2048
File system block size.....: 4096
Physical block size.....: 512
Device size in physical blocks...: 2097152
Building bootmap in '/boot'
Building menu 'zipl-automatic-menu'
Adding #1: IPL section 'Red Hat Enterprise Linux (4.18.0-305.e18.s390x) 8.4 (Ootpa)' (default)
  kernel image.....: /boot/secure-linux
  component address:
    heap area.....: 0x00002000-0x00005fff
    stack area.....: 0x0000f000-0x0000ffff
    internal loader.: 0x0000a000-0x0000dfff
    parameters.....: 0x00009000-0x000091ff
    kernel image....: 0x00010000-0x0229ffff
Adding #2: IPL section 'Red Hat Enterprise Linux
(0-rescue-9a84c2b961c340c492dcf817435f5a74) 8.4 (Ootpa)'
  initial ramdisk...:
  /boot/initramfs-0-rescue-9a84c2b961c340c492dcf817435f5a74.img
  kernel image.....: /boot/vmlinuz-0-rescue-9a84c2b961c340c492dcf817435f5a74
  kernel parmline...: 'root=/dev/mapper/rhel-root crashkernel=auto
rd.luks.uuid=luks-8898ca51-2780-4c6e-a959-75ad7ecb4eac rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap'
  component address:
    heap area.....: 0x00002000-0x00005fff
    stack area.....: 0x0000f000-0x0000ffff
    internal loader.: 0x0000a000-0x0000dfff
    parameters.....: 0x00009000-0x000091ff
    kernel image....: 0x00010000-0x006f1fff
    parmline.....: 0x006f3000-0x006f31ff
    initial ramdisk.: 0x00700000-0x0360dbff
Preparing boot device: vda (0000).
Detected SCSI PCBIOS disk layout.
Writing SCSI master boot record.
Syncing disks...
Done.
```

Figure 8-88 Running `zipl -V`

8.9.8 Securely removing the original unprotected kernel, initrd, and parmfile files

The original boot image, initial RAM image, and kernel parameter file are unprotected. Therefore, if they are not removed, VMs that have Secure Execution enabled can still be vulnerable to hacking attempts or sensitive data mining.

Figure 8-89 shows how to remove the vulnerabilities.

```
[root@kvm1secguest02 ~]# shred /boot/initramfs-4.18.0-305.el8.s390x.img
[root@kvm1secguest02 ~]# shred /boot/vmlinuz-4.18.0-305.el8.s390x
[root@kvm1secguest02 ~]# shred /home/root/parmfile
```

Figure 8-89 Removing files from boot image

8.9.9 Further RHEL guest hardening

To further harden this guest, we disabled the emergency and rescue shells. In Figure 8-90, you can see that we masked the systemd services for emergency and rescue.

```
[root@kvm1secguest02 ssh]# systemctl mask emergency.service
Created symlink /etc/systemd/system/emergency.service ? /dev/null.
[root@kvm1secguest02 ssh]# systemctl mask emergency.target
Created symlink /etc/systemd/system/emergency.target ? /dev/null.
[root@kvm1secguest02 ssh]# systemctl mask rescue.service
Created symlink /etc/systemd/system/rescue.service ? /dev/null.
[root@kvm1secguest02 ssh]# systemctl mask rescue.target
Created symlink /etc/systemd/system/rescue.target ? /dev/null
```

Figure 8-90 Disabling emergency and rescue shells

8.9.10 Removing unencrypted, older artifacts from /boot

The /boot partition is unencrypted. It is important to remove files that a malicious individual might use to attempt to start our virtual server without IBM Secure Execution.

Before removing files from /boot, it suggested to place a copy somewhere that is secure. For this purpose, we created a directory that is called /root/boot and copied the entire contents of /boot in to it. Keep in mind that the file system that /root is in is fully encrypted. In this way, if you want to regenerate secure-linux, the necessary input files are available.

Next, we use the srm program to securely erase files in /boot that are no longer needed. This process includes the initrd and vmlinuz files and related symbolic links.

In Example 8-23, you can see the final contents of the /boot file system. The only file in /boot contains the encryption and decryption key is secure-linux, and it is encrypted. The other files were securely deleted by using the srm program.

Example 8-23 Final /boot file system

```
root@secgquest02:/boot# ls -la
total 34284
drwxr-xr-x 3 root root 4096 Sep 8 15:56 .
drwxr-xr-x 19 root root 4096 Aug 3 13:33 ..
-rw----- 1 root root 43520 Sep 8 15:53 bootmap
-rw-r--r-- 1 root root 90430 Jul 9 19:50 config-5.4.0-42-generic
drwx----- 2 root root 16384 Aug 3 13:27 lost+found
-rw-r--r-- 1 root root 31911936 Sep 8 15:51 secure-linux
-rw----- 1 root root 3088846 Jul 9 19:50 System.map-5.4.0-42-generic
root@secgquest02:/boot#
```



IBM Cloud Infrastructure Center on Kernel-based Virtual Machines

This chapter provides information about installing and operating IBM Cloud Infrastructure Center on Kernel-based Virtual Machines (KVM).

This chapter includes the following topics:

- ▶ 9.1, “Installing IBM Cloud Infrastructure Center” on page 322
- ▶ 9.2, “Configuring IBM Cloud Infrastructure Center” on page 335
- ▶ 9.3, “Creating a bond for KVM administration network interfaces” on page 365

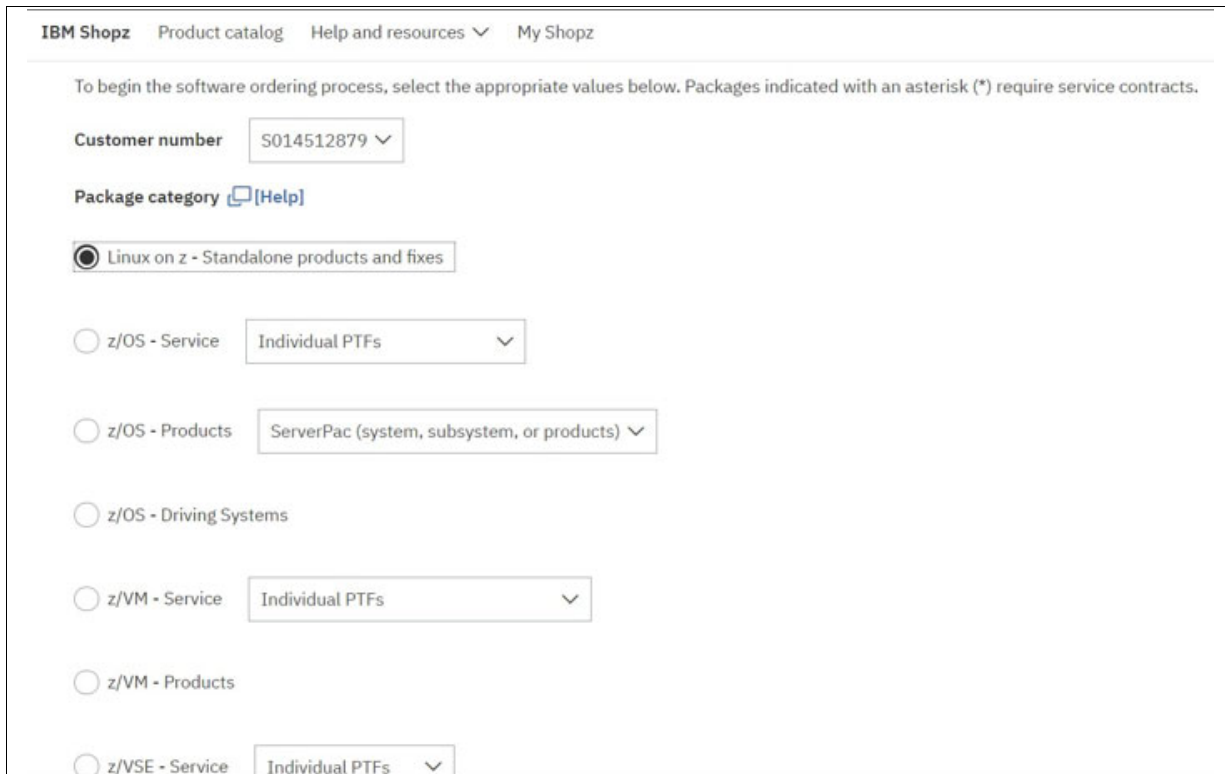
For more information about the hardware and software requirements for an IBM Cloud Infrastructure Center installation on KVM, see this [IBM Documentation web page](#).

9.1 Installing IBM Cloud Infrastructure Center

Complete the following steps to install IBM Cloud Infrastructure Center by using IBM ShopZ:

Note: You must use your customer number to purchase the software. In the ShopZ figures that are included here, our internal customer number is hidden.

1. Sign in to the [IBM Shopz web page](#) by using your customer account.
2. Search for “Linux on Z - Standalone products” and fixes (see Figure 9-1).



The screenshot shows the IBM ShopZ product selection interface. At the top, there are navigation links: "IBM Shopz", "Product catalog", "Help and resources" (with a dropdown arrow), and "My Shopz". Below this, a message states: "To begin the software ordering process, select the appropriate values below. Packages indicated with an asterisk (*) require service contracts." The form includes a "Customer number" field with the value "S014512879" and a dropdown arrow. Below that is a "Package category" section with a "[Help]" link. The selected category is "Linux on z - Standalone products and fixes", indicated by a radio button. Other categories listed include "z/OS - Service" (with a dropdown menu showing "Individual PTFs"), "z/OS - Products" (with a dropdown menu showing "ServerPac (system, subsystem, or products)"), "z/OS - Driving Systems", "z/VM - Service" (with a dropdown menu showing "Individual PTFs"), "z/VM - Products", and "z/VSE - Service" (with a dropdown menu showing "Individual PTFs").

Figure 9-1 IBM ShopZ Products

3. Create an order (see Figure 9-2).

Step 1 of 7 Specify order basics

Shopz **My orders** My preferences My hardware systems My licensed/installed software My downloads

Overview Create new order **My current order** Draft orders Processing Awaiting approval Completed

NOTE: For entitled standalone order, you could go to "My Downloads" to download it directly instead of placing an order.

Review and specify the basic details of your order.

Order name

Customer

Operating environment Linux on z

Package category Products

Package type [\[Help\]](#) Standalone products and fixes

Fast access to Shopz

Figure 9-2 Creating an order

4. Select the system on which the product is to run (see Figure 9-3).

Step 2 of 7 Select hardware systems

IBM CIC 1.1.4

Shopz **My orders** My preferences My hardware systems My licensed/installed software My downloads

Overview Create new order **My current order** Draft orders Processing Awaiting approval Completed

Select the system(s) on which you plan to run the software you are ordering.

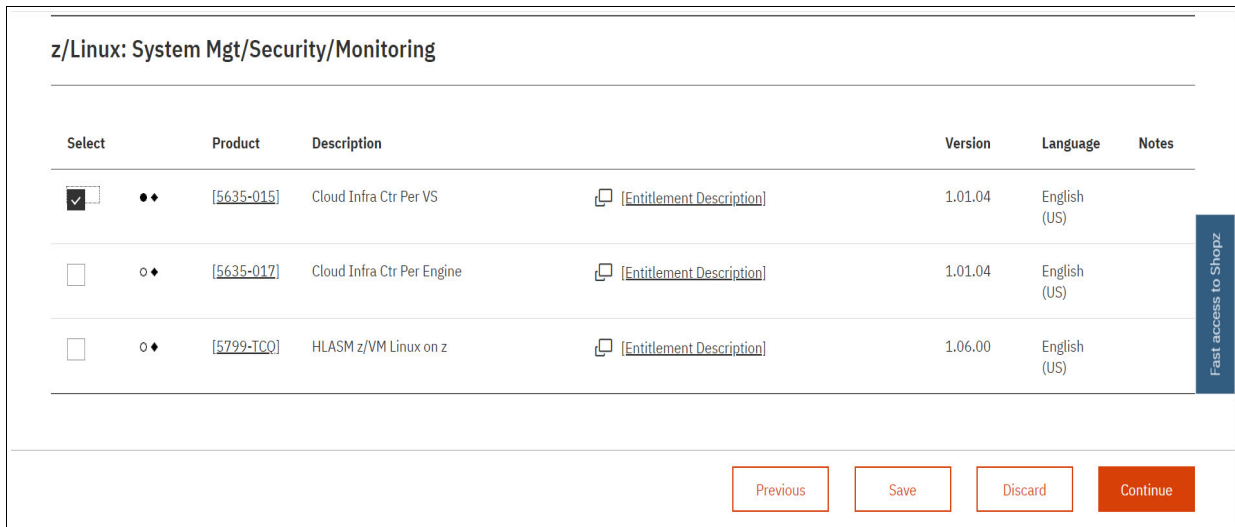
If you don't see your system listed, select the "unlisted" system and provide its details below.

Hardware systems for customer number

Group	Select	Designated machine	Description	Type-model	Serial no.
	<input type="checkbox"/>	2094022991E		2094-712	2991E

Figure 9-3 Selecting the system on which the product runs

The list of available products that are based on your search are shown (see Figure 9-4).



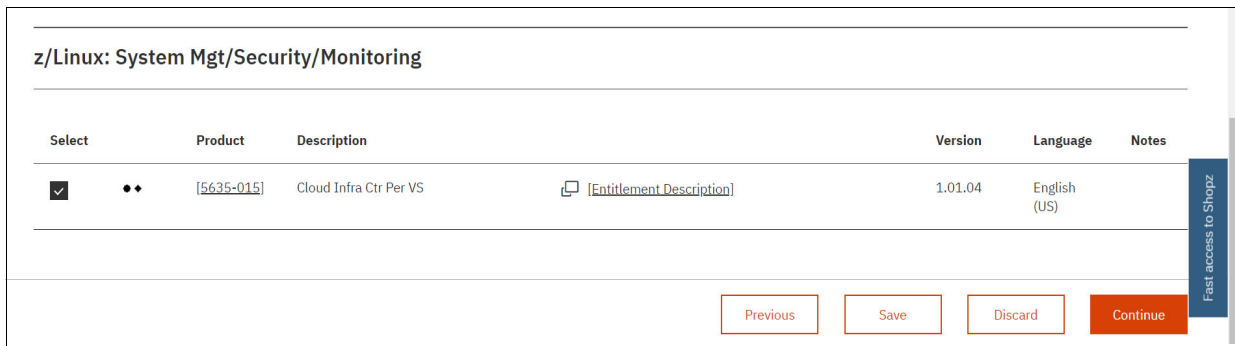
Select	Product	Description	Version	Language	Notes
<input checked="" type="checkbox"/>	♦♦ [5635-015]	Cloud Infra Ctr Per VS	1.01.04	English (US)	[Entitlement Description]
<input type="checkbox"/>	♦ [5635-017]	Cloud Infra Ctr Per Engine	1.01.04	English (US)	[Entitlement Description]
<input type="checkbox"/>	♦ [5799-TCO]	HLASM z/VM Linux on z	1.06.00	English (US)	[Entitlement Description]

Fast access to Shopz

Previous Save Discard Continue

Figure 9-4 List of available products based on your search

5. Select the package based on the license that you want to purchase (see Figure 9-5).



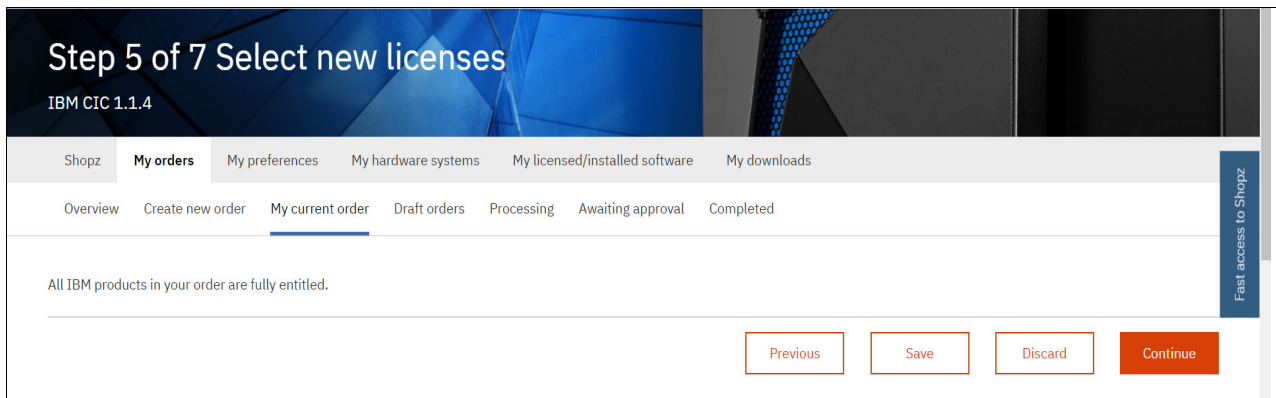
Select	Product	Description	Version	Language	Notes
<input checked="" type="checkbox"/>	♦♦ [5635-015]	Cloud Infra Ctr Per VS	1.01.04	English (US)	[Entitlement Description]

Fast access to Shopz

Previous Save Discard Continue

Figure 9-5 Selecting a package

6. Click **Continue** and you receive the output to validate the order (see Figure 9-6). Click **Continue**.



Step 5 of 7 Select new licenses

IBM CIC 1.1.4

Shopz **My orders** My preferences My hardware systems My licensed/installed software My downloads

Overview Create new order **My current order** Draft orders Processing Awaiting approval Completed

All IBM products in your order are fully entitled.

Fast access to Shopz

Previous Save Discard Continue

Figure 9-6 Selecting new licenses

7. Select the method that you want to use to receive the product; for example, by way of the internet. Click **Next** (see Figure 9-7).

Specify how you would like your order delivered.

ⓘ **ATTENTION:** If special instructions are added, the order processing time will be extended.

The fields indicated with an asterisk (*) are required to complete this transaction; other fields are optional. If you do not want to provide us with the required information, please use the "Back" button on your browser to return to the previous page, or close the window or browser session that is displaying this page.

Delivery media

Preferred media* >

Figure 9-7 Delivery media

Now, you can download the package to your machine (see Figure 9-8).

Product	Description	Version	Language	Delivery media
5635-015	Cloud Infra Ctr Per VS	1.01.04	English (US)	Internet

[← Edit Order contents](#)

New licenses
None

Figure 9-8 Order contents

8. Upload the downloaded file to the `/opt/icic_install` folder on the controller node (that is, `rdbkvm1.az12.dal.cpc.ibm.com`). Ensure that enough space is available to upload this file.

If the `/opt/icic_install` folder is not created, use the following command:

```
# mkdir -pv /opt/icic_install folder
mkdir: created directory '/opt/icic_install'
mkdir: created directory 'folder'
```

9. Upload file to the `/opt/icic_install` folder:

```
# scp IBM_Cloud_Infrastructure_Center_1.1.4.tar.gz
root@rdbkvm1.az12.dal.cpc.ibm.com:/opt/icic_install/
root@rdbkvm1.az12.dal.cpc.ibm.com's password:
IBM_Cloud_Infrastructure_Center_1.1.4.tar.gz
100% 345MB 368.2MB/s 00:00
```

10. Ensure that the required dnf repositories for IBM Cloud Infrastructure Center installation are working. Also, verify that dnf correctly added BaseOS, AppStream, and CodeReady repositories, as shown in Example 9-1.

Example 9-1 Verifying that dnf correctly added repositories

```
Figure 9-9 on page 337[root@rdbkvm1 ~]# dnf repolist
Updating Subscription Management repositories.
repo id
repo name
codeready-builder-for-rhel-8-s390x-rpms
Red Hat CodeReady Linux Builder for RHEL 8 IBM z Systems (RPMs)
rhel-8-for-s390x-appstream-rpms
Red Hat Enterprise Linux 8 for IBM z Systems - AppStream (RPMs)
rhel-8-for-s390x-baseos-rpms
Red Hat Enterprise Linux 8 for IBM z Systems - BaseOS (RPMs)
```

```
[root@rdbkvm2 ~]# dnf repolist
Updating Subscription Management repositories.
repo id
repo name
codeready-builder-for-rhel-8-s390x-rpms
Red Hat CodeReady Linux Builder for RHEL 8 IBM z Systems (RPMs)
rhel-8-for-s390x-appstream-rpms
Red Hat Enterprise Linux 8 for IBM z Systems - AppStream (RPMs)
rhel-8-for-s390x-baseos-rpms
Red Hat Enterprise Linux 8 for IBM z Systems - BaseOS (RPMs)
```

```
[root@rdbkvm3 ~]# dnf repolist
Updating Subscription Management repositories.
repo id
repo name
codeready-builder-for-rhel-8-s390x-rpms
Red Hat CodeReady Linux Builder for RHEL 8 IBM z Systems (RPMs)
rhel-8-for-s390x-appstream-rpms
Red Hat Enterprise Linux 8 for IBM z Systems - AppStream (RPMs)
rhel-8-for-s390x-baseos-rpms
Red Hat Enterprise Linux 8 for IBM z Systems - BaseOS (RPMs)
```

11. Use subscription-manager to list available repositories:

```
subscription-manager repos
```

In our KVM environment, we needed to enable several Red Hat repositories. Use subscription-manager if you need to enable them, as shown in Example 9-2.

Example 9-2 Enabling Red Hat repositories

```
# subscription-manager repos --list-enabled
+-----+
  Available Repositories in /etc/yum.repos.d/redhat.repo
+-----+
Repo ID:   rhel-8-for-s390x-baseos-rpms
Repo Name: Red Hat Enterprise Linux 8 for IBM z Systems - BaseOS (RPMs)
Repo URL:  https://cdn.redhat.com/content/dist/rhel8/$releasever/s390x/baseos/os
Enabled:   1

Repo ID:   rhel-8-for-s390x-appstream-rpms
```

Repo Name: Red Hat Enterprise Linux 8 for IBM z Systems - AppStream (RPMs)
Repo URL:
[https://cdn.redhat.com/content/dist/rhel8/\\$releasever/s390x/appstream/os](https://cdn.redhat.com/content/dist/rhel8/$releasever/s390x/appstream/os)
Enabled: 1

Repo ID: codeready-builder-for-rhel-8-s390x-rpms
Repo Name: Red Hat CodeReady Linux Builder for RHEL 8 IBM z Systems (RPMs)
Repo URL:
[https://cdn.redhat.com/content/dist/rhel8/\\$releasever/s390x/codeready-builder/os](https://cdn.redhat.com/content/dist/rhel8/$releasever/s390x/codeready-builder/os)
Enabled: 1

12. Configure the firewall settings on the controller, as shown in Example 9-3.

Example 9-3 Configuring firewall settings

```
# systemctl status firewalld
? firewalld.service - firewalld - dynamic firewall daemon
  Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled; vendor
  preset: enabled)
  Active: active (running) since Thu 2021-11-25 17:53:26 UTC; 3 days ago
  Docs: man:firewalld(1)
  Main PID: 1643 (firewalld)
  Tasks: 2 (limit: 3355442)
  Memory: 32.7M
  CGroup: /system.slice/firewalld.service
          ??1643 /usr/libexec/platform-python -s /usr/sbin/firewalld --nofork
  --nopic
```

```
Nov 25 17:53:26 rdbkvm1.az12.dal.cpc.ibm.com systemd[1]: Starting firewalld -
dynamic firewall daemon...
Nov 25 17:53:26 rdbkvm1.az12.dal.cpc.ibm.com systemd[1]: Started firewalld -
dynamic firewall daemon.
Nov 25 17:53:26 rdbkvm1.az12.dal.cpc.ibm.com firewalld[1643]: W
```

```
[root@rdbkvm1 ~]# firewall-cmd --list-all
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: encb80
  sources:
  services: cockpit dhcpv6-client ssh
  ports:
  protocols:
  forward: no
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

13. Open the firewall ports for IBM Cloud Infrastructure Center.

Note: For more information about the firewall ports that are used by IBM Cloud Infrastructure Center, see this [IBM Documentation web page](#).

The outbound ports are not required to be configured in the local firewall, as shown in Example 9-4.

Example 9-4 Outbound ports that are not required to be configured in local firewall

```
[root@rdbkvm1 ~]# firewall-cmd --permanent
--add-port={80/tcp,443/tcp,5000/tcp,5470/tcp,5671/tcp,8080/tcp,8428/tcp,8774/tcp,8770/tcp,8998/tcp,9000/tcp,9292/tcp,9696/tcp,35357/tcp,8041/tcp,8778/tcp,4369/tcp,9191/tcp,8775/tcp,50110/tcp,6200/tcp,6201/tcp,6202/tcp,6080/tcp}
success

[root@rdbkvm1 ~]# firewall-cmd --reload
success

[root@rdbkvm1 ~]# firewall-cmd --list-all
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: encb80
  sources:
  services: cockpit dhcpv6-client ssh
ports: 80/tcp 443/tcp 5000/tcp 5470/tcp 5671/tcp 8080/tcp 8428/tcp 8774/tcp
8770/tcp 8998/tcp 9000/tcp 9292/tcp 9696/tcp 35357/tcp 8041/tcp 8778/tcp 4369/tcp
9191/tcp 8775/tcp 50110/tcp 6200/tcp 6201/tcp 6202/tcp 6080/tcp
  protocols:
  forward: no
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

14. Confirm that KVM controller (rdbkvm1) can ping both KVM compute nodes, as shown in Example 9-5.

Example 9-5 Confirming KVM controller ping

```
# ping -c2 rdbkvm2.az12.dal.cpc.ibm.com
PING rdbkvm2.az12.dal.cpc.ibm.com (9.214.220.202) 56(84) bytes of data.
64 bytes from rdbkvm2.az12.dal.cpc.ibm.com (9.214.220.202): icmp_seq=1 ttl=64
time=0.269 ms
64 bytes from rdbkvm2.az12.dal.cpc.ibm.com (9.214.220.202): icmp_seq=2 ttl=64
time=0.079 ms

--- rdbkvm2.az12.dal.cpc.ibm.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1039ms
rtt min/avg/max/mdev = 0.079/0.174/0.269/0.095 ms

# ping -c2 rdbkvm3.az12.dal.cpc.ibm.com
PING rdbkvm3.az12.dal.cpc.ibm.com (9.214.220.201) 56(84) bytes of data.
```



```
64 bytes from rdbkkvm3.az12.dal.cpc.ibm.com (9.214.220.201): icmp_seq=1 ttl=64
time=0.193 ms
64 bytes from rdbkkvm3.az12.dal.cpc.ibm.com (9.214.220.201): icmp_seq=2 ttl=64
time=0.098 ms
```

```
--- rdbkkvm3.az12.dal.cpc.ibm.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1010ms
rtt min/avg/max/mdev = 0.098/0.145/0.193/0.049 ms
```

Note: If the `nc` package is not installed, run the `yum install nmap-ncat` command.

15. Check whether the controller node can access KVM compute nodes on port 22/tcp, as shown in Example 9-6.

Example 9-6 Confirming whether controller node can access KVM compute nodes

```
# nc -z -v -w 1 rdbkkvm2.az12.dal.cpc.ibm.com 22
Ncat: Version 7.70 ( https://nmap.org/ncat )
Ncat: Connected to 9.214.220.202:22.
Ncat: 0 bytes sent, 0 bytes received in 0.01 seconds.

# nc -z -v -w 1 rdbkkvm3.az12.dal.cpc.ibm.com 22
Ncat: Version 7.70 ( https://nmap.org/ncat )
Ncat: Connected to 9.214.220.201:22.
Ncat: 0 bytes sent, 0 bytes received in 0.01 seconds.
```

16. Set up the IBM Cloud Infrastructure Center configuration file.

Before installing IBM Cloud Infrastructure Center, create a directory that is named `icic` in `/etc`, and create a file within it that is named `config.properties`. This `/etc/icic/config.properties` file should include configurations that are customized to your deployment environments, as shown in the following configuration properties example (be sure to specify only one property):

```
[icic configs]
compute_instance_template=ins%05x
```

The virtual machine (VM) user IDs that are generated by the `compute_instance_template` must be 8 characters. For example, when the value is specified as `ins%05x`, the VM user IDs are generated with `ins` as the prefix and followed by the index `ins00000`, `ins00001` - `insfffff` (ensure that no conflict exists between the VM to be created and existing VMs), which means the total VM capacity is 1048576.

Carefully determine the `compute_instance_template` value to ensure sufficient VM capacity because you cannot change `compute_instance_template` after the management node is installed.

Note: The installation fails if one the following conditions exist:

- ▶ The `config.properties` file is missing.
- ▶ The `compute_instance_template` property is missing.
- ▶ The generated VM user IDs length is not 8.
- ▶ Any non-alphanumeric characters are in the prefix.
- ▶ Less than 4 bytes are used for the index.

Create a directory to install IBM Cloud Infrastructure Center by using the following commands:

```
[root@rdbkvm1 ~]# mkdir -pv /etc/icic
mkdir: created directory '/etc/icic'

[root@rdbkvm1 ~]# cat /etc/icic/config.properties
[icic configs]
compute_instance_template=ins%05x
```

9.1.1 Before you install IBM Cloud Infrastructure Center

Complete the following steps before installing:

1. Check the operating system Red Hat version on all nodes, as shown in Example 9-7.

Example 9-7 Checking Red Hat version on all nodes

Controller:

```
[root@rdbkvm1 ~]# cat /etc/os-release
NAME="Red Hat Enterprise Linux"
VERSION="8.4 (Ootpa)"
ID="rhel"
ID_LIKE="fedora"
VERSION_ID="8.4"
PLATFORM_ID="platform:el8"
PRETTY_NAME="Red Hat Enterprise Linux 8.4 (Ootpa)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:redhat:enterprise_linux:8.4:GA"
HOME_URL="https://www.redhat.com/"
DOCUMENTATION_URL="https://access.redhat.com/documentation/red_hat_enterprise_linu
x/8/"
BUG_REPORT_URL="https://bugzilla.redhat.com/"

REDHAT_BUGZILLA_PRODUCT="Red Hat Enterprise Linux 8"
REDHAT_BUGZILLA_PRODUCT_VERSION=8.4
REDHAT_SUPPORT_PRODUCT="Red Hat Enterprise Linux"
REDHAT_SUPPORT_PRODUCT_VERSION="8.4"
Red Hat Enterprise Linux release 8.4 (Ootpa)
Red Hat Enterprise Linux release 8.4 (Ootpa)
```

Compute nodes:

```
[root@rdbkvm2 ~]# cat /etc/os-release
NAME="Red Hat Enterprise Linux"
VERSION="8.4 (Ootpa)"
ID="rhel"
ID_LIKE="fedora"
VERSION_ID="8.4"
PLATFORM_ID="platform:el8"
PRETTY_NAME="Red Hat Enterprise Linux 8.4 (Ootpa)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:redhat:enterprise_linux:8.4:GA"
HOME_URL="https://www.redhat.com/"
DOCUMENTATION_URL="https://access.redhat.com/documentation/red_hat_enterprise_linu
x/8/"
BUG_REPORT_URL="https://bugzilla.redhat.com/"
```

```

REDHAT_BUGZILLA_PRODUCT="Red Hat Enterprise Linux 8"
REDHAT_BUGZILLA_PRODUCT_VERSION=8.4
REDHAT_SUPPORT_PRODUCT="Red Hat Enterprise Linux"
REDHAT_SUPPORT_PRODUCT_VERSION="8.4"
Red Hat Enterprise Linux release 8.4 (Ootpa)
Red Hat Enterprise Linux release 8.4 (Ootpa)

[root@rdbkvm3 ~]# cat /etc/os-release
NAME="Red Hat Enterprise Linux"
VERSION="8.4 (Ootpa)"
ID="rhel"
ID_LIKE="fedora"
VERSION_ID="8.4"
PLATFORM_ID="platform:el8"
PRETTY_NAME="Red Hat Enterprise Linux 8.4 (Ootpa)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:redhat:enterprise_linux:8.4:GA"
HOME_URL="https://www.redhat.com/"
DOCUMENTATION_URL="https://access.redhat.com/documentation/red_hat_enterprise_linu
x/8/"
BUG_REPORT_URL="https://bugzilla.redhat.com/"

REDHAT_BUGZILLA_PRODUCT="Red Hat Enterprise Linux 8"
REDHAT_BUGZILLA_PRODUCT_VERSION=8.4
REDHAT_SUPPORT_PRODUCT="Red Hat Enterprise Linux"
REDHAT_SUPPORT_PRODUCT_VERSION="8.4"
Red Hat Enterprise Linux release 8.4 (Ootpa)
Red Hat Enterprise Linux release 8.4 (Ootpa)

```

By default, IBM Cloud Infrastructure Center uses eth0 for its network interface (for more information, see this [IBM Documentation web page](#)).

IBM Cloud Infrastructure Center on a KVM management node requires a dedicated network interface for communicating with compute nodes. If you want to use eth0 for communicating with compute nodes, or if you want to use a different network interface to host the web UI for any other reason, you can set the environment variable `HOST_INTERFACE` before running the installation script; for example, export `HOST_INTERFACE=eth1`.

In our environment, all Red Hat Linux servers were configured with the encb80 as the NIC interface, as shown in Example 9-8.

Example 9-8 Red Hat Linux servers configured with the encb80 NIC interface

```

# lsqeth
Device name                : encb80
-----
card_type                   : OSD_25GIG
cdev0                       : 0.0.0b80
cdev1                       : 0.0.0b81
cdev2                       : 0.0.0b82
chpid                       : B8
online                      : 1
portname                   : no portname required
portno                     : 0

```

```
state                : UP (LAN ONLINE)
priority_queueing    : disabled
buffer_count         : 64
layer2               : 1
isolation            : none
bridge_role          : none
bridge_state         : inactive
bridge_hostnotify    : 0
bridge_reflect_promisc : none
switch_attrs        : unknown
vnicc/flooding       : 0
vnicc/learning       : 0
vnicc/learning_timeout : 600
vnicc/mcast_flooding : 0
vnicc/rx_bcast       : 1
vnicc/takeover_learning : 0
```

IBM Cloud Infrastructure Center requires the en_US.UTF-8 locale. Ensure that the command locale can be run without any warning, and that the locale environment variables are set to en_US.UTF-8, as shown in the following example:

```
# locale
LANG=en_US.UTF-8
LC_CTYPE="en_US.UTF-8"
LC_NUMERIC="en_US.UTF-8"
LC_TIME="en_US.UTF-8"
LC_COLLATE="en_US.UTF-8"
LC_MONETARY="en_US.UTF-8"
LC_MESSAGES="en_US.UTF-8"
LC_PAPER="en_US.UTF-8"
LC_NAME="en_US.UTF-8"
LC_ADDRESS="en_US.UTF-8"
LC_TELEPHONE="en_US.UTF-8"
LC_MEASUREMENT="en_US.UTF-8"
LC_IDENTIFICATION="en_US.UTF-8"
LC_ALL=
```

Default values of LC_CTYPE=en_US.UTF-8 and LANG=en_US.UTF-8 are used if these variables are not set before the installation.

Note: Changing the management node's hostname after IBM Cloud Infrastructure Center is installed can cause the service to become unavailable or operations to time out. Ensure that you set your hostname *before* IBM Cloud Infrastructure Center is installed.

2. Ensure that the correct locale setting is defined by using the following command on the controller node:

```
# export LC_CTYPE=en_US.UTF-8
```

3. Unpack the installation components. The process varies depending on which type of installer package was downloaded.

Note: Before unpacking, ensure that you obtained an installer that matches your licensing model. As of Version 1.1.4.0, IBM Cloud Infrastructure Center is offered with Per Virtual Server or Per Engine licensing.

4. Complete the following steps to extract the installation files:

- a. Switch to the `/opt/icic_install` directory:

```
# cd /opt/icic_install/
```

- b. Extract the installation files:

```
# tar zxvfp IBM_Cloud_Infrastructure_Center_1.1.4.tar.gz
icic-install-s390x-rhel-1.1.4.0.tgz
icic-install-1.1.4.0.tgz.sig
icicpublickey
readme.txt
```

After you mount the ISO or extract the installation package, the following files are available:

- `icic-install-s390x-rhel-1.1.4.0.tgz`
- `icic-install-1.1.4.0.tgz.sig`
- `icicpublickey`
- `readme.txt`

- c. Run the following command to extract the `icic-install-s390x-rhel-1.1.4.0.tgz` compressed binary file:

```
# tar -xzvf icic-install-s390x-rhel-1.1.4.0.tgz
```

- d. Change directory to `/opt/icic_install/icic-1.1.4.0`:

```
# cd /opt/icic_install/icic-1.1.4.0
```

- e. Export the `HOST_INTERFACE` variable. Remember to use the interface name that was collected previously (in our case, `enclb80`):

```
# export HOST_INTERFACE=enclb80
```

5. Incorporate the following required SELinux settings:

```
# ./install -s -k
#####
Starting the IBM Cloud Infrastructure Center 1.1.4.0 installation on:
2021-12-02T16:49:35+00:00
#####
```

Note: SELINUX is not enforced. Enable and enforce SELINUX and then, attempt to install again. The installation exits with error code 266.

To enable it, edit the `/etc/selinux/config` file by using `vim` editor and update the following parameter:

```
SELINUX=enforcing
```

- Restart your controller node after this update completes. After server is running, export the variables:

```
# export HOST_INTERFACE=enclb80
# export LC_CTYPE=en_US.UTF-8
```

- Install the management node.

The installer verifies that the environment meets all of the prerequisites during execution. If any errors are reported during the verification steps of the installation process, correct the reported issues and attempt the installation again.

As a root user, start the installation by using one of the following methods:

- Run the following command to start the installation:

```
./install -k
```

You are prompted to respond to some questions during the installation.

- Press **Enter** to continue viewing the license agreement, or, enter one of the following choices:

- 1 to accept the agreement
- 2 to decline the agreement
- 3 to print the agreement
- 4 to read non-IBM terms
- 99 to return to the previous window

For our example, we choose the **1 to accept the agreement** option.

- You are prompted to decide whether you want the IBM Cloud Infrastructure Center setup to configure the firewall (1 for Yes, 2 for No). For our example, we choose **2**.

Note: The firewall service is disabled because the iptables service is used by IBM Cloud Infrastructure Center.

- You are prompted whether you want to continue with the installation (1 for Yes, 2 for No). For our example, we choose **1**.

- Run the following command to start a silent installation:

```
/install -s -k
```

Note: The `-c` option must be passed for the installer to configure firewall settings automatically during a silent installation. Otherwise, you must manually set up the firewall rules. The `-c` option disables and stops the firewall first and then, enables, configures, and starts iptables.

You should receive the following output when the installation completes:

```
The validation of IBM Cloud Infrastructure Center services post install was
successful.
```

```
*****
IBM Cloud Infrastructure Center installation successfully completed at
2021-12-02T18:41:25+00:00.
```

For more information, see the
`/opt/ibm/icic/log/icic_install_2021-12-02-183208.log` file.

The following URL is used to access IBM Cloud Infrastructure Center:
`https://9.214.220.203` (replace this URL with your own server IP)

Firewall configuration might be required to use IBM Cloud Infrastructure Center.

Note: The installation log file is available at: `/opt/ibm/icic/log`.

8. After the installation completes, run the following command to ensure that the status of services is active:

```
/opt/ibm/icic/bin/icic-services status
```

If the installation fails, see this [IBM Documentation web page](#) to determine whether the problem you encountered is a known issue.

During the installation of the IBM Cloud Infrastructure Center, the `allow_lun_scan` on the management node is turned off, as shown in following output:

```
# cat /sys/module/zfcp/parameters/allow_lun_scan
```

If any volumes are mapped to your management node with the `allow_lun_scan` disabled, the system restart causes a volume loss because other kernel no longer performs a LUN scan automatically. Refer to FCP LUNs that are part of the root file system or FCP LUNs that are not part of the root file system to add and persist the volume.

For information about Performing a standard RHEL 8 installation, see this [Red Hat installation documentation web page](#).

For more information about IBM Cloud Infrastructure Center installation, see this [IBM Documentation web page](#).

9.2 Configuring IBM Cloud Infrastructure Center

Before adding the KVM compute nodes into IBM Cloud Infrastructure Center, ensure that the network OSA devices are active *before* OpenVswitch is enabled. Also, you must define the OSA devices to be a bridge port, which allows it to act as a member of a Linux software bridge.

Note: Use the `bridge_role` attribute of a network device in Layer 2 so that it can receive all traffic with unknown destination MAC addresses. For more information, see this [IBM Documentation web page](#).

This process that is described next must be completed or the compute nodes (KVM LPAR) cannot be added into IBM Cloud Infrastructure Center.

Complete the following steps to confirm that the two OSA cards are available on the KVM compute nodes:

1. Connect to the KVM compute host and issue the following commands to set b00 and b10:
 - Make devices visible in Linux:

```
cio_ignore -r b00-b02
cio_ignore -r b10-b12
```

- Check whether devices are visible:
`lscss | grep <dev_number>`
- 2. Create the configuration for b00 and b10 network adapters:
`znetconf -A`
 Scanning for network devices...
 Successfully configured device 0.0.0b00 (encb00)
 Successfully configured device 0.0.0b10 (encb10)
- 3. Update the OSA configuration to support the bridge:
`lszdev 0.0.0b00 --columns ATTR:bridge_state`
`lszdev 0.0.0b10 --columns ATTR:bridge_state`
- 4. Disable zDev channels 0b00 and 0b10:
`chzdev --disable qeth 0.0.0b00`
`chzdev --disable qeth 0.0.0b10`
`chzdev qeth 0.0.0b10 layer2=1 bridge_role=primary`
`chzdev qeth 0.0.0b00 layer2=1 bridge_role=primary`
`chzdev -a 0.0.0b10 bridge_role=primary`
`chzdev -a 0.0.0b00 bridge_role=primary`
`chzdev --enable qeth 0.0.0b10`
`chzdev --enable qeth 0.0.0b00`
`lszdev 0.0.0b00 --columns ATTR:bridge_role`
`lszdev 0.0.0b10 --columns ATTR:bridge_role`

9.2.1 Other tasks

Access the UI to perform the following tasks:

- ▶ Add Hosts (Compute nodes)
- ▶ Add Networks
- ▶ Add Images
- ▶ Add Storage Providers

These tasks are described next.

Adding hosts (compute nodes)

Complete following steps to add the KVM hosts to IBM Cloud Infrastructure Center:

1. Access the UI at <https://9.214.220.203> (replace this URL with your own server IP), as shown in Figure 9-9.

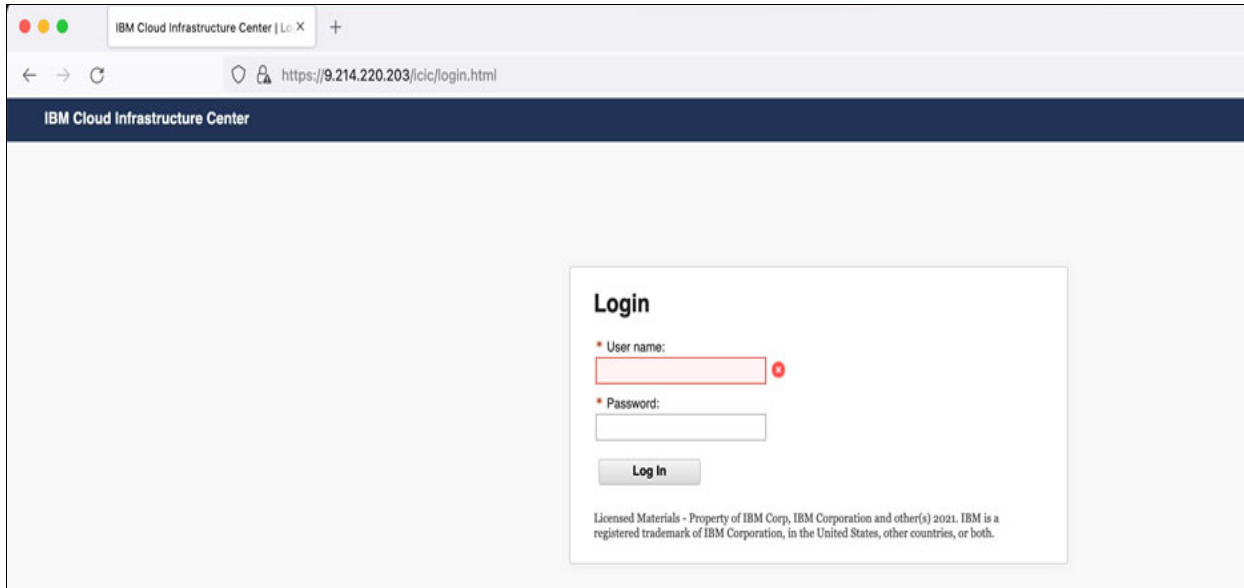


Figure 9-9 Log-in window

2. Log in by using the root ID and password (see Figure 9-10).

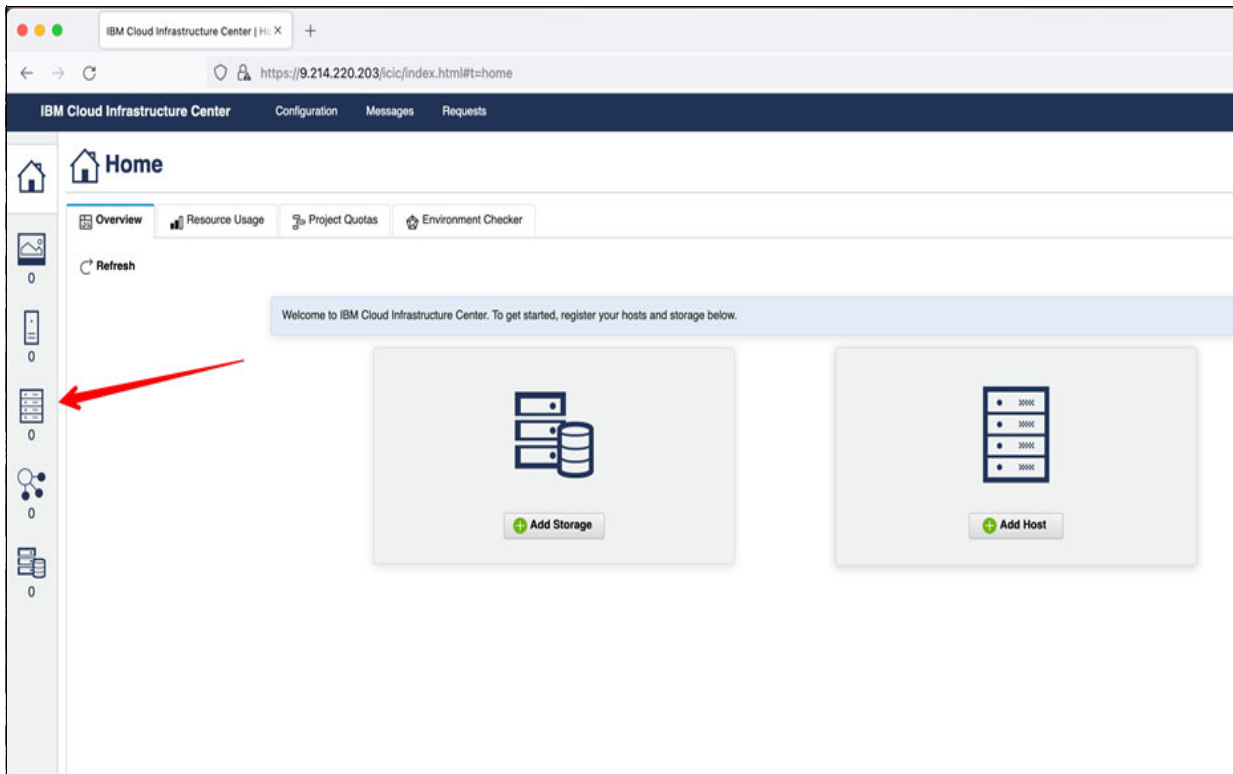


Figure 9-10 Adding Host

3. Click in the **Hosts** section (see Figure 9-11).

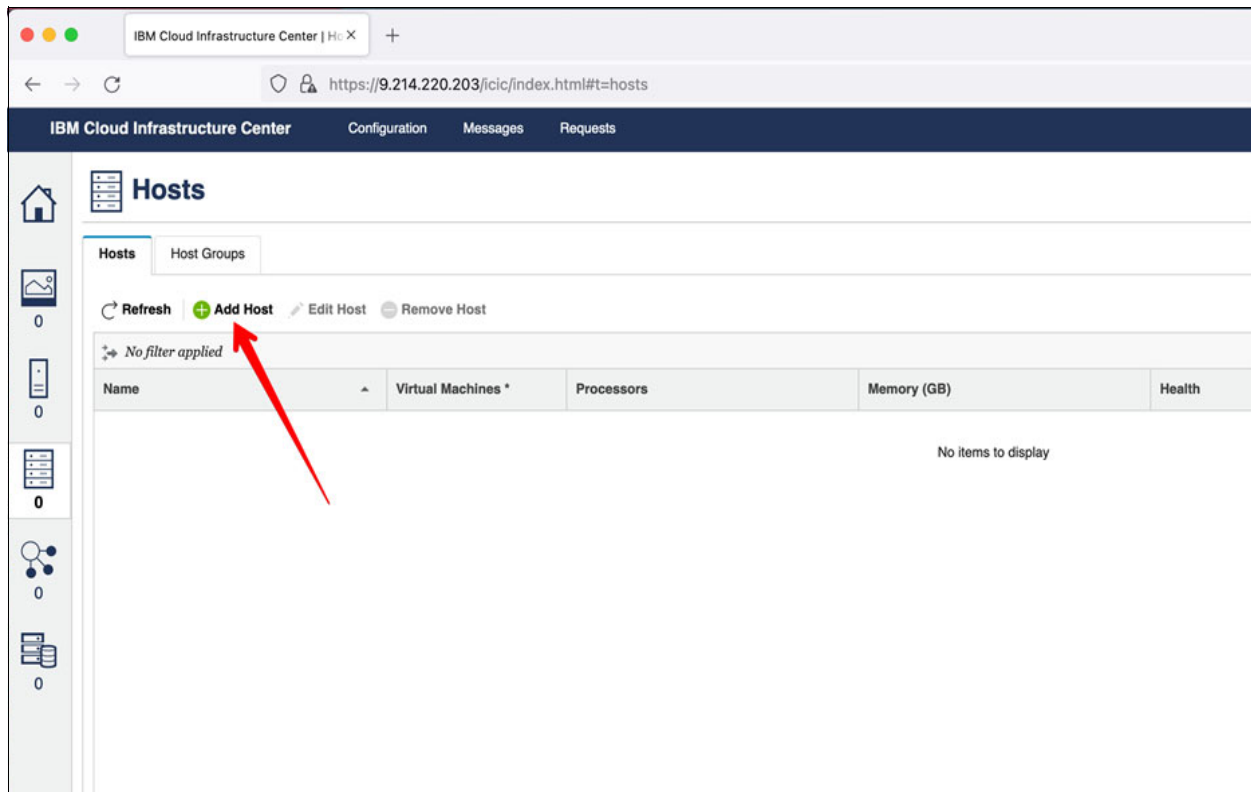


Figure 9-11 Add Host option

Figure 9-12 on page 339 - Figure 9-19 on page 343 show the sequence that is used to add and connect to a host, and log in to and define the host networks.

4. Click **Add Host** to add the first compute node (see Figure 9-11).

5. Enter the information about your compute node (see Figure 9-12).

Add Host

Specify the details for host registration.

Host management type:
 KVM

* Hostname or IP address: * User ID:


Display name: Authentication type:
 Password SSH key

* Network Interface name: * Password:

Shared storage path:

Figure 9-12 Add host details

6. Click **Connect** when you receive the dialog message shown in Figure 9-13.

 The authenticity of the KVM cannot be verified.

Details:
Host key fingerprint: ssh-ed25519 256
ca:75:51:55:07:9a:02:57:6a:54:9a:4c:20:51:34:a8

Are you sure you want to continue connecting?

Figure 9-13 Connecting to host

After several minutes, your compute node is added to the host section, as shown in Figure 9-14.

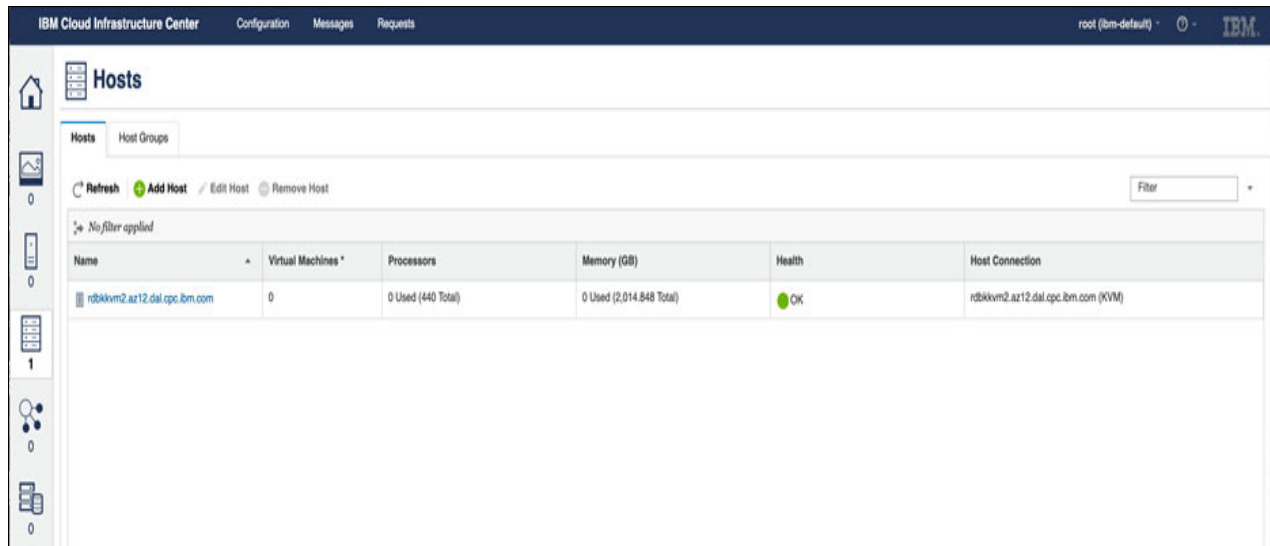


Figure 9-14 Host added

7. Complete the following steps to configure the OpenVswitch on rdbkvm2 and rdbkvm3 to use LACP and VLAN tagging:

Note: Replace the values that are in **bold** to match your network infrastructure.

- a. Delete the existing IBM Cloud Infrastructure Center definitions for default:


```
ovs-vsctl del-br default
```
- b. Create the new definitions by using LACP:


```
ovs-vsctl add-br default0# Add Virtual Switch Bond: VLANs 201, 214
ovs-vsctl add-bond default0 bond0 encb00 encb10 bond_mode=balance-tcp
other_config:bond-detect-mode=miimon other_config:bond-miimon-interval=100
other_config:trunks=201,214 other_config:bond_updelay=10
other_config:lacp-time=fast other-config:stp_enable=false
```
- c. Trunk VLANs 201, 214:


```
ovs-vsctl setport bond0 trunks=201,214 tag=1 vlan_mode=native-untagged
```
- d. Set LACP as active:


```
ovs-vsctl setport bond0 lacp=active
```
- e. Restart openvswitch:


```
stemctl restart openvswitch
```
- f. Validate the bond configuration:


```
ovs-vsctl list port
ovs-appctl bond/show bond0
ovs-appctl lacp/show
```

Repeat these steps on the other compute node (rdbkvm3).

Complete the following steps add networks to IBM Cloud Infrastructure Center:

1. Access the UI at <https://9.214.220.203>, as shown in Figure 9-15.

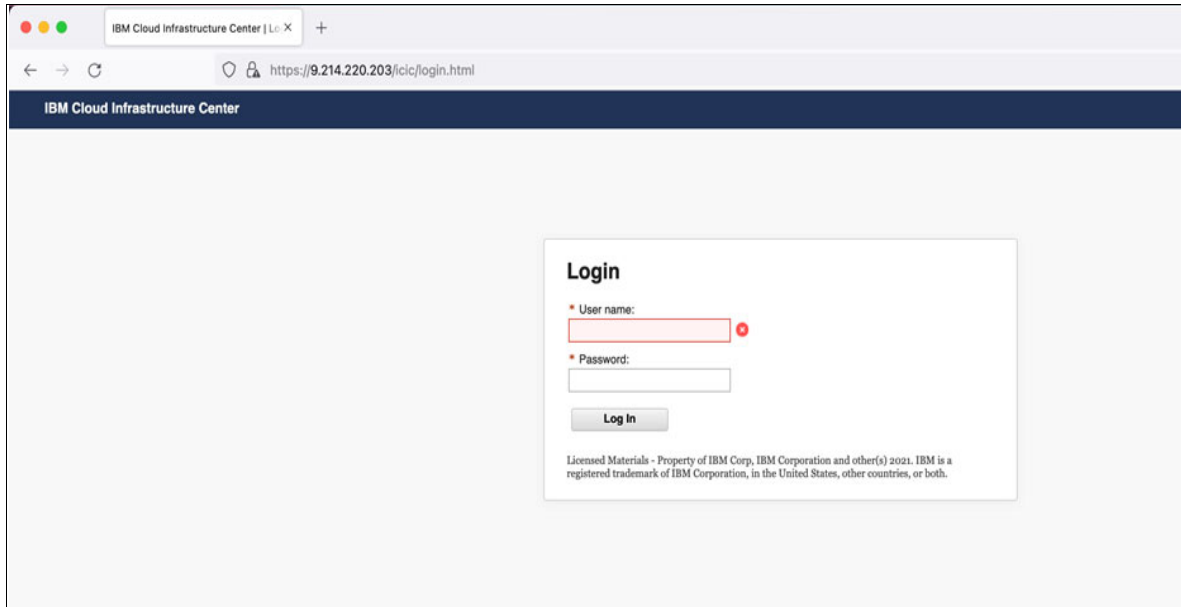


Figure 9-15 Host Login window

2. Log in by using the root ID and password.
3. Click the **Networks** icon, as shown in Figure 9-16.

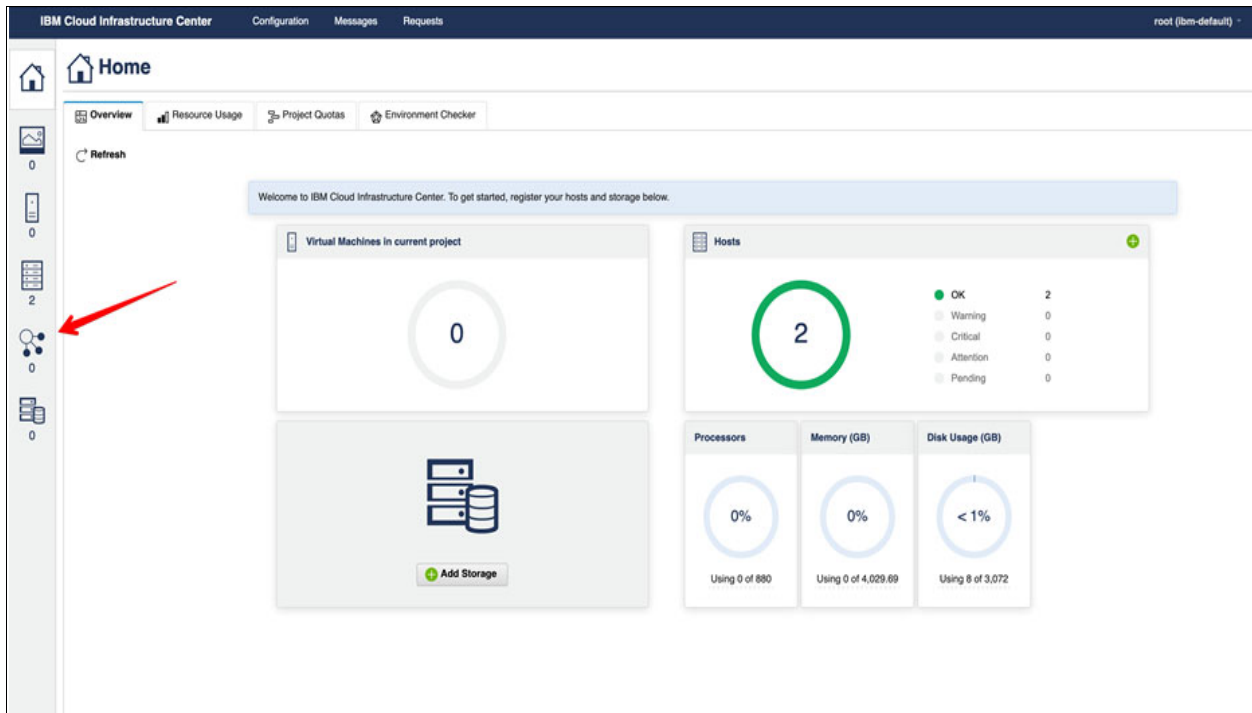


Figure 9-16 Selecting the Networks icon

4. Click **Add Network** (see Figure 9-17).

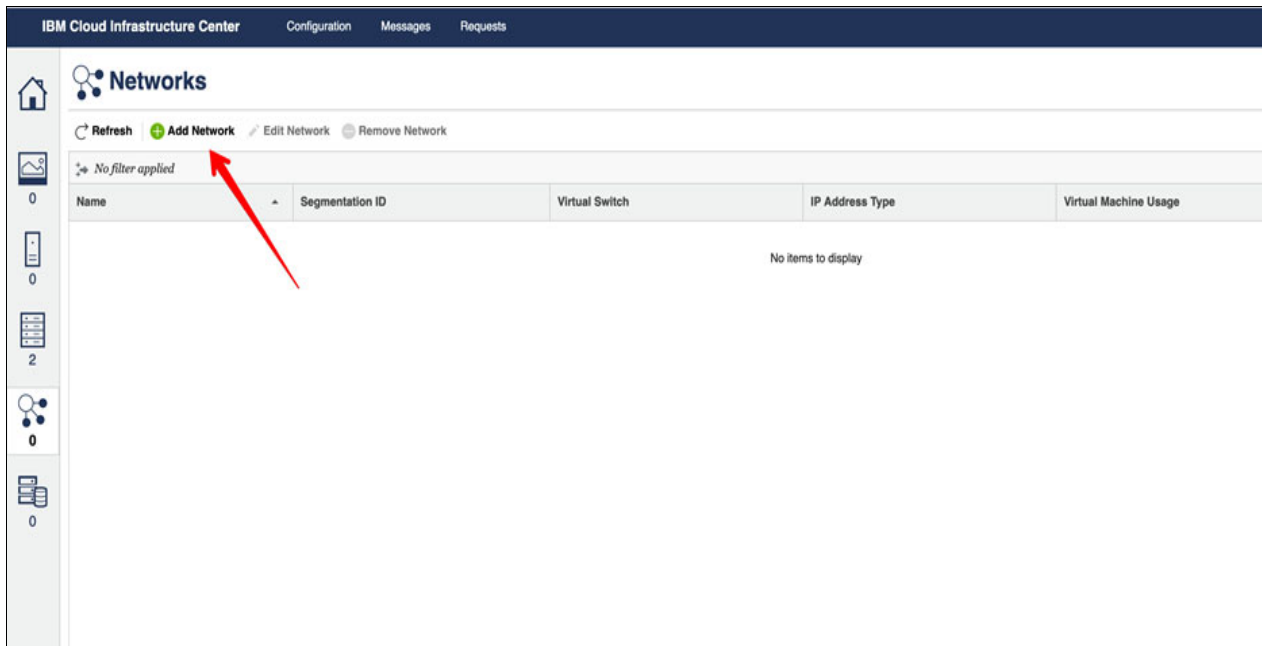


Figure 9-17 Selecting the Add Network option

5. Enter the information about your network subnet, as shown in Figure 9-18.

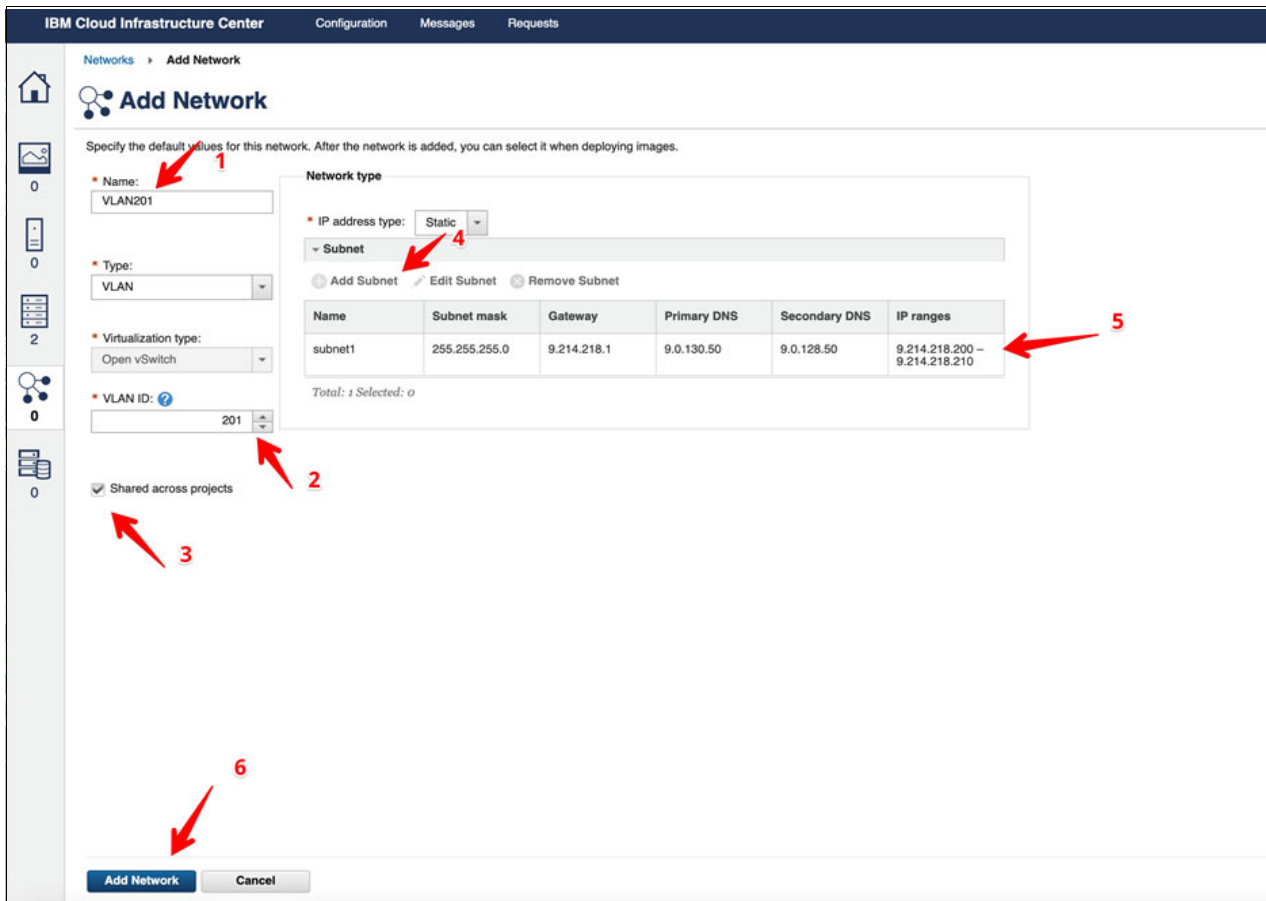


Figure 9-18 Adding network information and then, clicking the Add Network option

6. Repeat these steps to include all other network subnets. After the subnets are added, your networks are listed as shown in Figure 9-19.

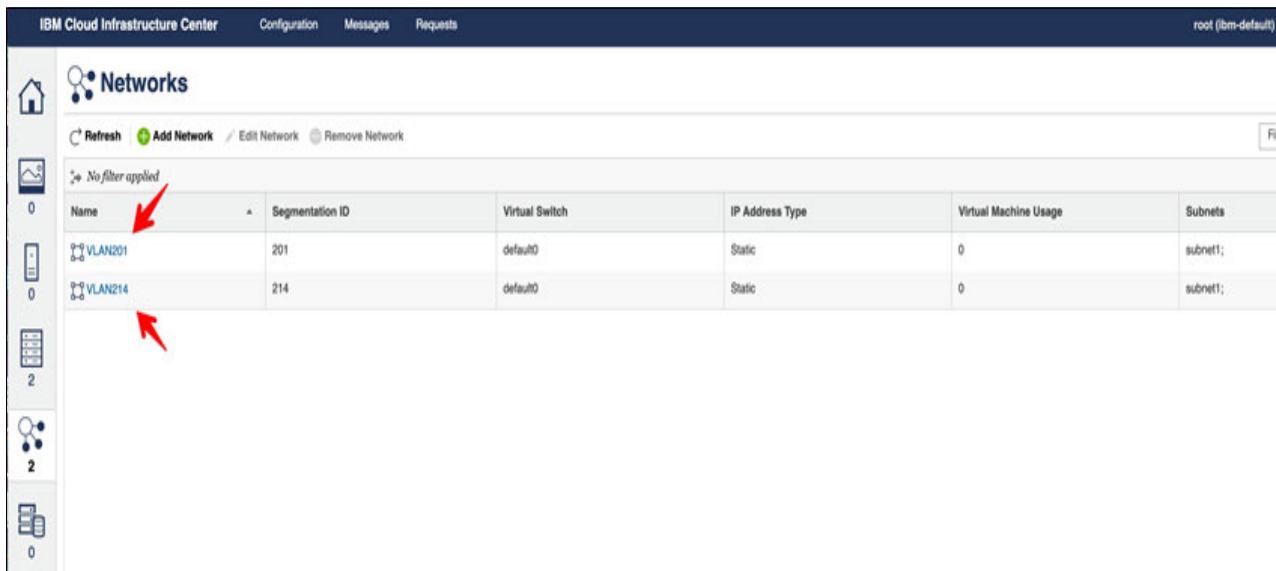


Figure 9-19 Networks added

9.2.2 Creating and adding images

The steps that are described in this section, we provide a quick method to create a Red Hat 8.4 image for IBM Cloud Infrastructure Center by way of the Red Hat Kickstart process.

For more information about Red Hat Kickstart, see this [Red Hat Documentation web page](#).

Complete the following steps on only the first KVM compute node (rdbkvm2):

1. Run the following command to create an encrypted password (it is used later to update the kickstart file):

```
python -c 'import crypt,getpass;pw=getpass.getpass();print(crypt.crypt(pw) if (pw==getpass.getpass("Confirm: ")) else exit()'
```

Note: When you are prompted to enter the initial password that is used for root ID, make a note of the encrypted password because it is used later.

2. Create /gpfs/img folder:

```
mkdir -pv /gpfs/img
```

3. Upload the Red Hat DVD image. (If you do not have the image, download it from this [Red Hat web page](#)):

```
ls -la /gpfs/img/rhel-8.4-s390x-dvd.iso
```

```
-rw-r--r--. 1 root root 3783262208 Dec  3 11:14 /gpfs/img/rhel-8.4-s390x-dvd.iso
```

4. Create the /gpfs/rhel-8.4-s390x-CLONE.cfg file by using the contents that is shown in Example 9-9. You must update the settings that are highlighted in **bold** with the encrypted root password that was created in Step 1 and update the IP address of the new virtual server.

Example 9-9 Configuration file contents

```
text
lang en_US.UTF-8
keyboard us
rootpw
$6$y9kVIF0tWVCHn3J0$qPP9arf1Ah1VRB4BJCbXALMYrKGJGSzD.6xGfUSnFAmB9cyUbrD7e4d56y
T.L1fC52ptHatKMxrmhT20x/eM1 --iscrypted
timezone America/New_York
poweroff
zerombr
ignoredisk --only-use=vda
clearpart --all

part /boot --fstype=xfs --size=1024 --asprimary

part pv.0 --fstype=lvm pv --size=8192 --grow
volgroup systemvg --pesize=8192 pv.0
logvol / --vgname=systemvg --name=root --fstype=xfs --size=1024
```



```
logvol /usr --vgname=systemvg --name=usr --fstype=xfst --size=4096
logvol /home --vgname=systemvg --name=home --fstype=xfst --size=128
logvol /opt --vgname=systemvg --name=opt --fstype=xfst --size=128
logvol /tmp --vgname=systemvg --name=tmp --fstype=xfst --size=128
logvol /var --vgname=systemvg --name=var --fstype=xfst --size=1024
logvol swap --vgname=systemvg --name=swap --fstype=swap --size=1024
```

```
syspurpose --role="Red Hat Enterprise Linux Server" --sla="Standard"
--usage="Development/Test"
```

```
auth --passalgo=sha512
selinux --permissive
firewall --disabled
skipx
firstboot --disable
%packages
@^server-product-environment
kexec-tools
bind-utils
device-mapper-multipath
e2fsprogs
git
iproute
ksh
net-tools
cloud-init
cloud-utils-growpart
procps
unzip
vim
zip
file
perl-Net-Ping
binutils
wget
rsync
postfix
gettext
yum
%end
```

```

%post
    subscription-manager unregister
    touch /etc/sudo.env
    cat /dev/null > /etc/multipath/bindings
    cat /dev/null > /etc/multipath/wwids
    cat /usr/share/doc/device-mapper-multipath-0.4.9/multipath.conf > /etc/multipath.conf
sed -i -e '/^users:$/,/^ - default$/d' -e 's/^disable_root: 1/disable_root: 0/' -e
's/^ssh_pwauth: 0/ssh_pwauth: 1/' /etc/cloud/cloud.cfg
    echo "Installation completed."
%end

```

5. Export BASH variables to help create the VMs:

```

export ISO="/gpfs/img/rhel-8.4-s390x-dvd.iso"
export MachineName='rhe184_kvm'

```

ISO contains the path for the Red Hat 8.4 DVD .iso file, which you must download). The MachineName parameter includes the name of the VM that is created in IBM Cloud Infrastructure Center.

6. Create the QCOW2 file. In the following command, a 10 GB QCOW2 file is created for our root disk. We decided to use 10 GB because this size is the minimum disk size for the default IBM Cloud Infrastructure Center compute templates:

```

qemu-img create -f qcow2 -o preallocation=metadata /gpfs/${MachineName}.qcow2
10G

```

7. Ensure that the virt-install package is installed:

```

yum install -y virt-install

```

8. Run the following command to automatically create the Red Hat VM:

```

virt-install --virt-type=kvm --name=${MachineName} --vcpus=2 --memory=4096
--location="${ISO}" --disk path=/gpfs/${MachineName}.qcow2,format=qcow2
--network default --noreboot --initrd-inject=/gpfs/rhel-8.4-s390x-CLONE.cfg
--extra-args "inst.ks=file:/rhel-8.4-s390x-CLONE.cfg"

```

Note: the installation takes some minutes to complete. Wait until the process is completed. You should receive the output that is shown in Figure 9-20.

Figure 9-20 shows the messages that indicate that the installation completed.

```
Verifying vim-filesystem.noarch (672/676)
Verifying volume_key-libs.s390x (673/676)
Verifying wget.s390x (674/676)
Verifying xdg-utils.noarch (675/676)
Verifying xkeyboard-config.noarch (676/676)
.
Installing boot loader
..
Performing post-installation setup tasks
.
[terminated]
```




Figure 9-20 Installation completed message

9. After Kickstart completes the Linux installation, a Red Hat VM with 10 GB is available. Run the following command to compress this image. Doing so reduces the image size from 10 GB to a much smaller size (it is always better to work with small images files):

```
qemu-img convert -c -O qcow2 /gpfs/${MachineName}.qcow2
/gpfs/${MachineName}_compressed.qcow2
```

After converting the image, you can verify the size of the new compressed file. The file size should be reduced from 10 GB to approximately 1.1 GB.

```
du -hs /gpfs/${MachineName}_compressed.qcow2
1.1G/gpfs/rhel84_kvm_compressed.qcow2
```

10. Upload the compressed file to /opt/icic_install on KVM controller server (rdbkvm1):

```
# scp /gpfs/${MachineName}_compressed.qcow2 root@rdbkvm1:/opt/icic_install
root@rdbkvm1's password:
rhel84_kvm_compressed.qcow2
100% 1038MB 826.2MB/s 00:01
```

11. Connect to KVM controller server (rdbkvm1) and issue the following commands:

- a. Authenticate to IBM Cloud Infrastructure Center and provide the root password:

```
# source /opt/ibm/icic/icirc root
```

Enter the password for root:

- b. Import the Red Hat 8.4 image into IBM Cloud Infrastructure Center by using the following commands:

```
cd /opt/icic_install
```

```
openstack image create --min-disk 10 --file rhel84_kvm_compressed.qcow2
rhel-84-kvm-10g
```

```
openstack image set --property image_type_xcat=linux --property
hypervisor_type=kvm --property architecture=s390x --property os_name=Linux
--property os_version=rhel84 rhel-84-kvm-10g
```

```
openstack image set --property os_distro='Rhel8' --property os_version=8
rhel-84-kvm-10g
```

```
openstack image set --shared rhel-84-kvm-10g
```

```
openstack image set --public rhel-84-kvm-10g
```

- c. Return to UI and verify that the new imported image is listed in Images section (see Figure 9-21).

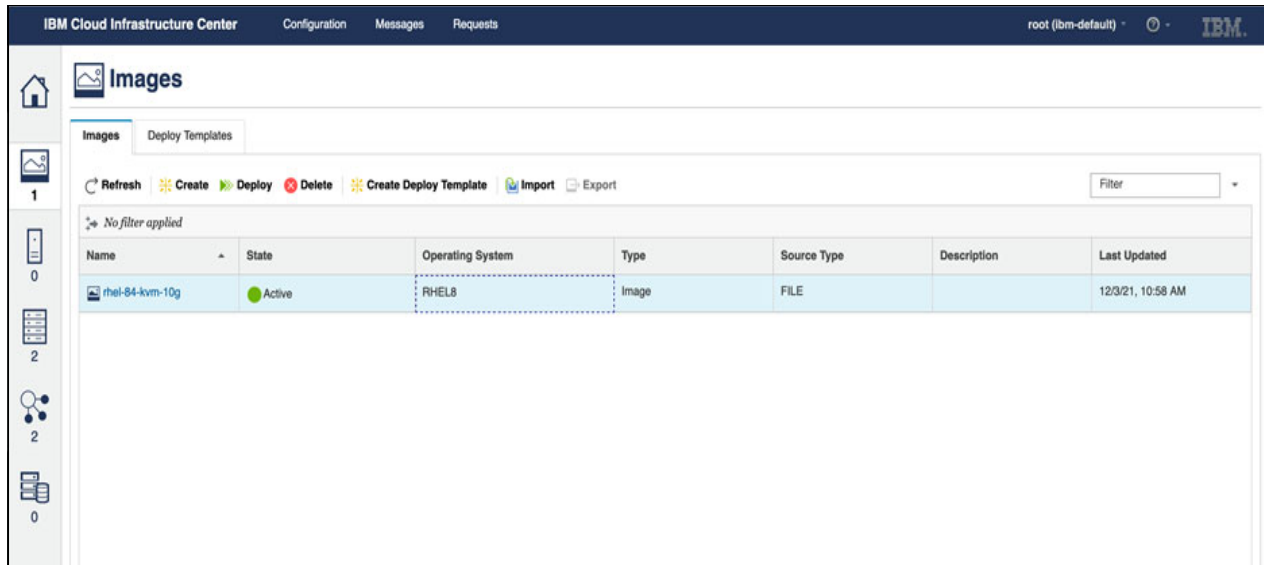


Figure 9-21 Adding Images

Although the image can be imported by using UI, it can take longer than the process that was described here if your internet connection is slow. The reason is because you must download the Red Hat compressed file to your machine and then, upload it to the KVM controller node.

Alternatively, you can manually perform the Red Hat traditional installation by completing the following steps:

1. Export some BASH variables to help create the VM:

```
export ISO="/gpfs/img/rhel-8.4-s390x-dvd.iso"
export MachineName='rhel84_kvm'
```

ISO contains the path of the Red Hat 8.4 DVD .iso file. The MachineName parameter includes the name of the VM that is created in KVM.

2. Create the QCOW2 file for the VM by using the following command:

```
qemu-img create -f qcow2 -o preallocation=metadata /gpfs/${MachineName}.qcow2 10G
```

3. Ensure that the virt-install package is installed:

```
yum install -y virt-install
```

4. Run the following command to automatically create the Red Hat VM:

```
virt-install --name=${MachineName} --vcpus=2 --memory=4096 --disk path=/gpfs/${MachineName}.qcow2,format=qcow2 --import --network network:VLAN214 --cdrom="${ISO}"
```

The installer starts and you are prompted to respond several questions. When the installation is complete, go to the next step.

5. Repeat steps 9 and 10 on page 346 to compress and import the Red Hat image into IBM Cloud Infrastructure Center.

9.2.3 Adding storage providers

Complete the following steps to add an IBM DS8K storage to IBM Cloud Infrastructure Center. The provided storage allows you to attach and manage SAN disks for the Linux VMs:

1. Ensure that you are logged in to IBM Cloud Infrastructure Center with a privileged user (that is, root) in the UI.
2. Click in the Storage Providers section (see Figure 9-22).

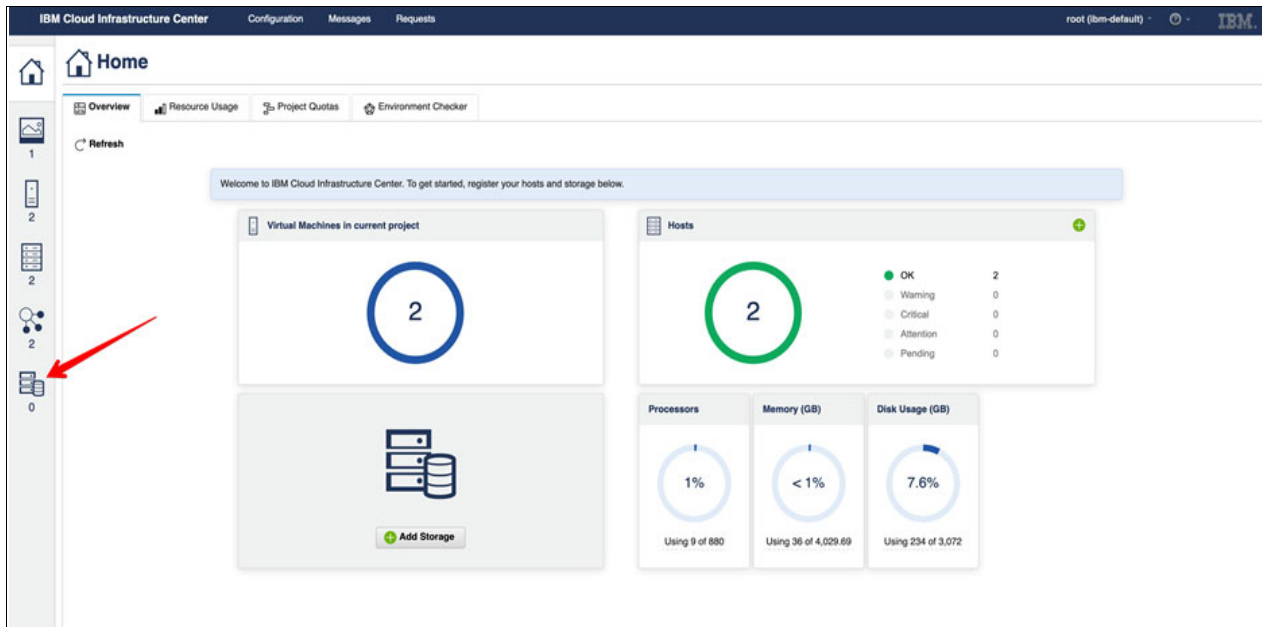


Figure 9-22 Storage providers

3. Click **Add Storage** (see Figure 9-23).

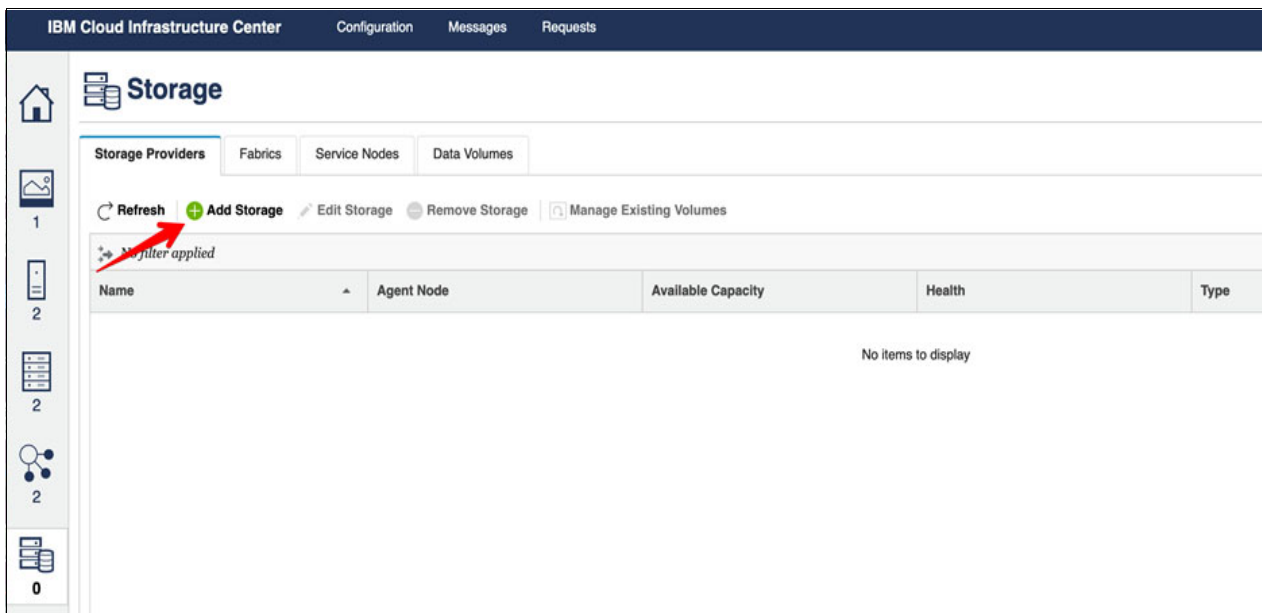


Figure 9-23 Add Storage option

4. Complete the information about the added storage and click **Connect** (see Figure 9-24).

Add Storage
For each new storage provider, a default storage template is created. You can modify the template after the storage provider has been added.

Specify a storage controller

* Agent Node

* Availability zone

* Type:

* Hostname or IP address:

* User ID:

* Display name:

* Password:

Connect

Add Storage **Cancel**

Figure 9-24 Storage added details

5. Click **Accept** to accept the IBM DS8000 SSL certificate (see Figure 9-25).

The certificate for the IBM DS8000 is untrusted or invalid.

Details:

Subject: D5LBT80.az12.dal.cpc.ibm.com
Issuer: HMC Root CA
Issued on: August 3, 2020 at 4:28:06 PM Horário Padrão de Brasília
Expires on: July 12, 2030 at 4:28:06 PM Horário Padrão de Brasília
SHA1 fingerprint: 1c:86:b0:46:cc:21:45:0:af:d1:f3:7a:fa:22:f9:fc:2d:8d:3e:c3

Are you sure you want to continue connecting?

Connect **Cancel**

Figure 9-25 Connecting to storage

- Select the storage pool that is to be used as default and then, click **Add Storage** (see Figure 9-26).

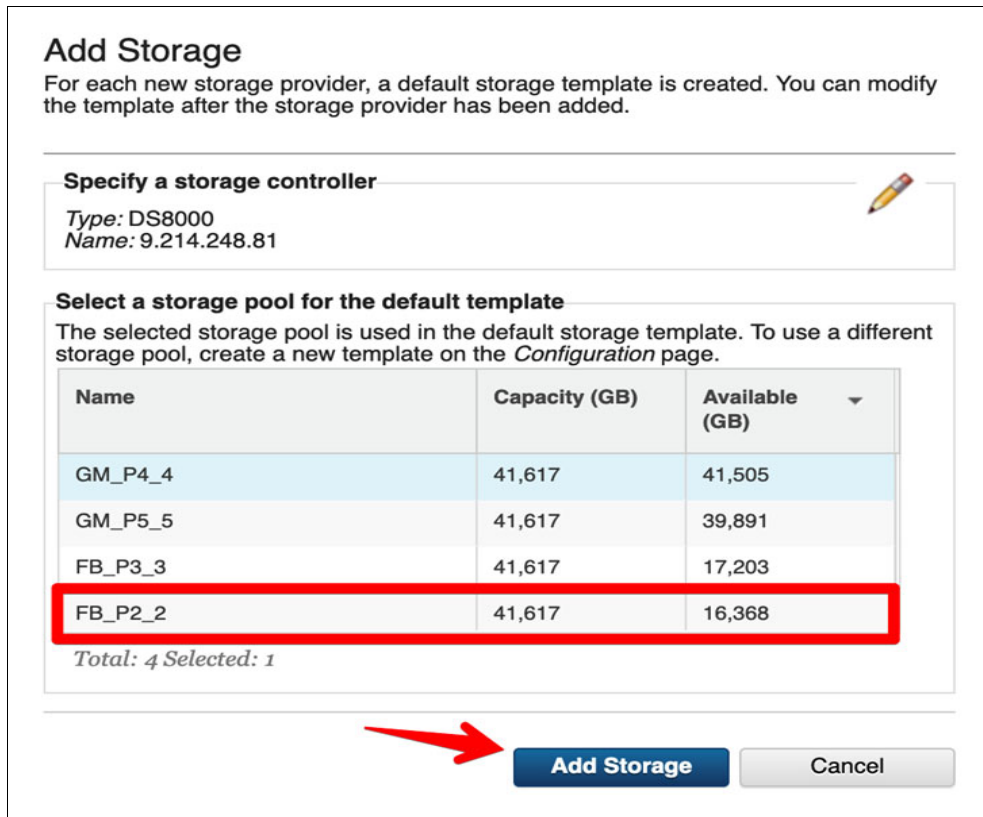


Figure 9-26 Adding storage

- After the process completes, the DS800 storage is listed in the Storage Providers section, as shown in Figure 9-27.

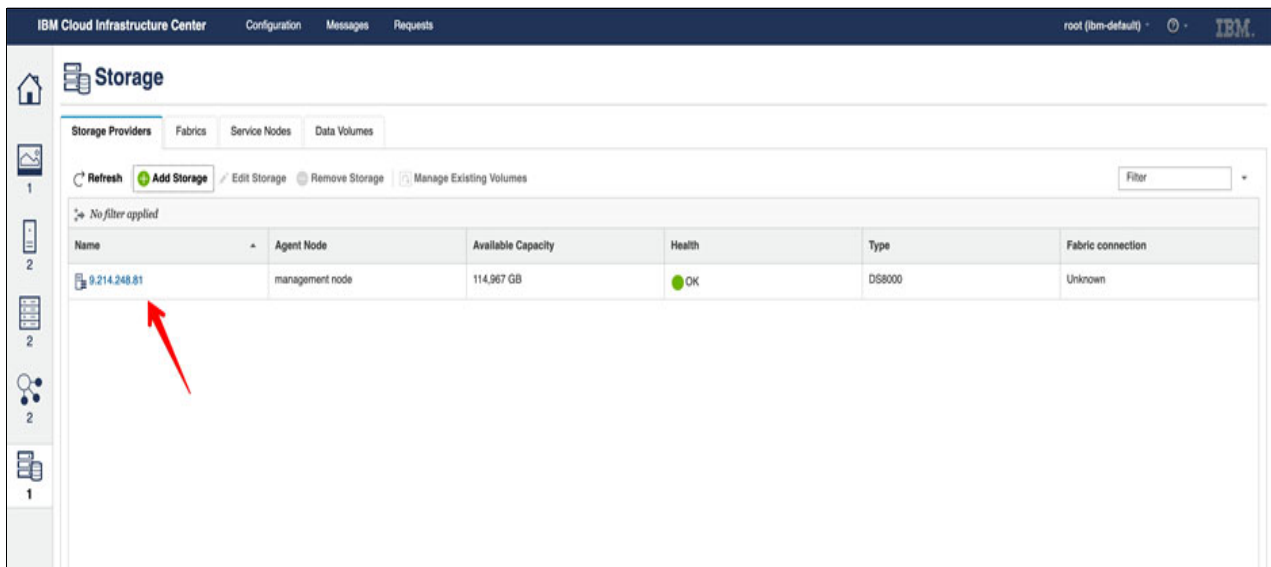
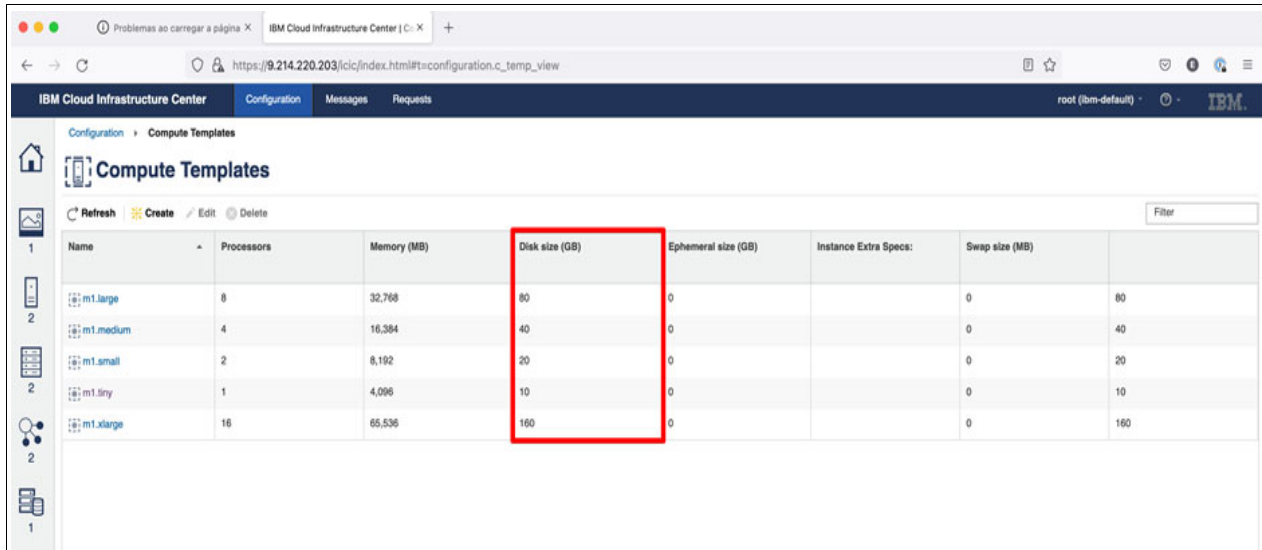


Figure 9-27 Storage is added

9.2.4 Extending the root file system

Initially, the Red Hat Linux image that was created in the previous section was defined with a 10 GB disk. Therefore, any VM that is created with a compute template (t-shirt size) that is not tiny requires that more space is made available.

The IBM Cloud Infrastructure Center installation defined the default compute templates that are shown in Figure 9-28. The disk size is highlighted to show only tiny has 10 GB that is defined in disk size. Any other compute template that is selected to provision a new VM requires manual steps to make the added space available.



Name	Processors	Memory (MB)	Disk size (GB)	Ephemeral size (GB)	Instance Extra Specs:	Swap size (MB)
m1.large	8	32,768	80	0		0
m1.medium	4	16,384	40	0		0
m1.small	2	8,192	20	0		0
m1.tiny	1	4,096	10	0		0
m1.xlarge	16	65,536	160	0		0

Figure 9-28 Sizing storage

Complete the following steps to simulate a required resize:

1. Build a large VM.
2. Connected to the VM by using the `lsblk` command. Notice that the VM total disk capacity is now 80 GB (it was 10 GB):

```
]# lsblk
NAME                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0                  11:0    1  476K  0 rom
vda                  252:0    0   80G  0 disk
├─vda1                252:1    0    1G  0 part /boot
└─vda2                252:2    0    9G  0 part
   ├─systemvg-root    253:0    0    1G  0 lvm /
   ├─systemvg-swap    253:1    0    1G  0 lvm [SWAP]
   ├─systemvg-usr     253:2    0    4G  0 lvm /usr
   ├─systemvg-var     253:3    0    1G  0 lvm /var
   ├─systemvg-tmp     253:4    0  128M  0 lvm /tmp
   ├─systemvg-opt     253:5    0  128M  0 lvm /opt
   └─systemvg-home    253:6    0  128M  0 lvm /home
```


3. Complete the following steps to extend the operating system partition, refer to the next steps:

a. Run the **growpart** command to extend the vda partition 2:

```
# growpart /dev/vda 2
CHANGED: partition=2 start=2099200 old: size=18872320 end=20971520 new:
size=165672927 end
```

b. Resize the physical volume

```
# pvscan
PV /dev/vda2   VG systemvg      lvm2 [8.99 GiB / <1.62 GiB free]
Total: 1 [8.99 GiB] / in use: 1 [8.99 GiB] / in no VG: 0 [0  ]
# pvresize /dev/vda2
Physical volume "/dev/vda2" changed
1 physical volume(s) resized or updated / 0 physical volume(s) not resized
# pvscan
PV /dev/vda2   VG systemvg      lvm2 [78.99 GiB / <71.62 GiB free]
Total: 1 [78.99 GiB] / in use: 1 [78.99 GiB] / in no VG: 0 [0  ]
```

Now, LVM commands can be used to extend any LVM logical volume.

9.2.5 User tasks

If you are assigned the administrator role to a project in IBM Cloud Infrastructure Center, in addition to starting, stopping, restarting, and deleting a VM, you can resize or capture a VM, or attach or detach a volume to it. Other user roles might not be allowed to do these operations.

You also can perform a live migration, cold migration, or a suspend/pause/resume operation to a KVM VM. Users can perform the following tasks, which are described next:

- ▶ Create VMs
- ▶ Live migrate VMs
- ▶ Resize Virtual Machines
- ▶ Attach volumes to VMs
- ▶ Detach volumes to VMs
- ▶ Delete volumes
- ▶ Delete VMs

Creating virtual machines

You can use a Deploy Template to create a VM. Complete the following steps:

1. Log in to the suitable project and open the Images window (see Figure 9-29).

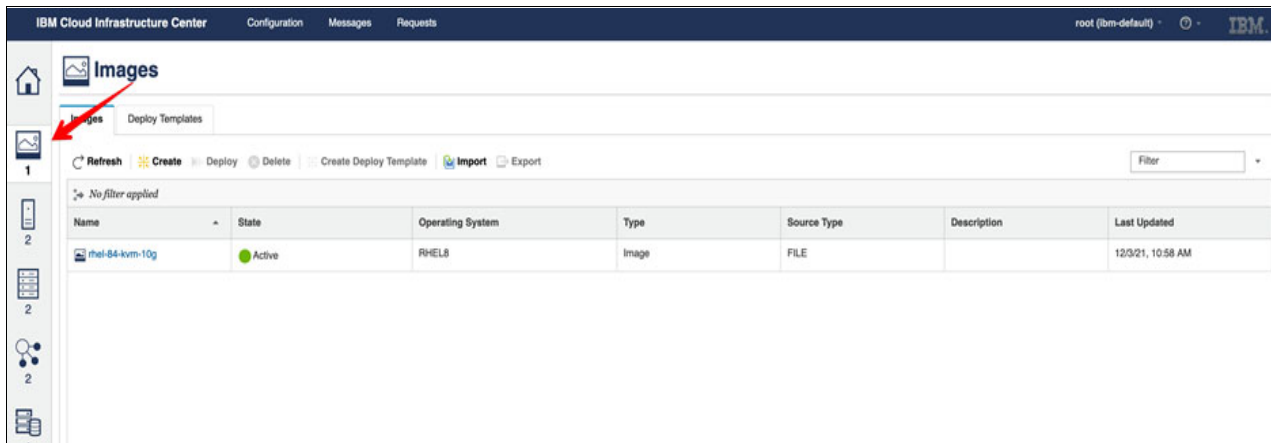


Figure 9-29 Creating an image

2. Select the image that you want to use and click **Deploy** (see Figure 9-30).

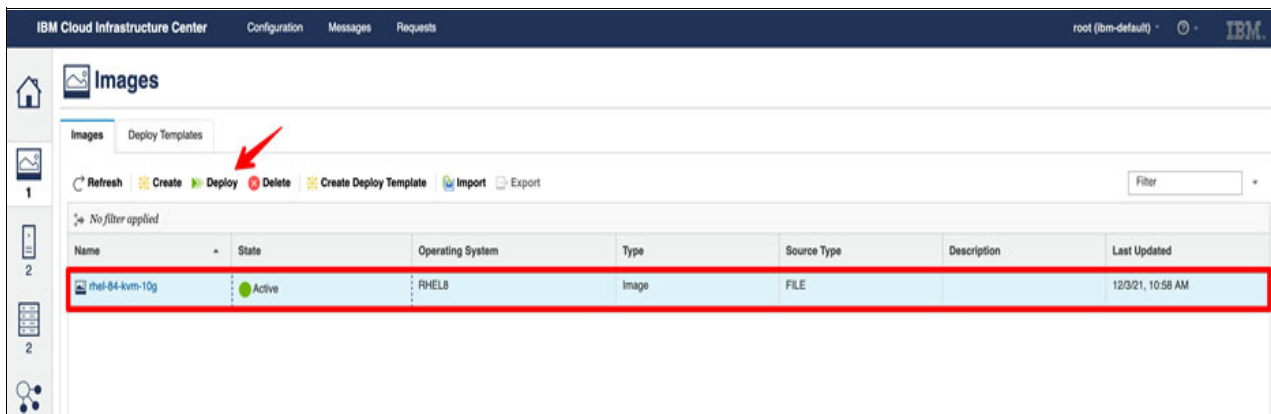


Figure 9-30 Deploying an image

3. Specify the requested details in the window that opens (see Figure 9-31).

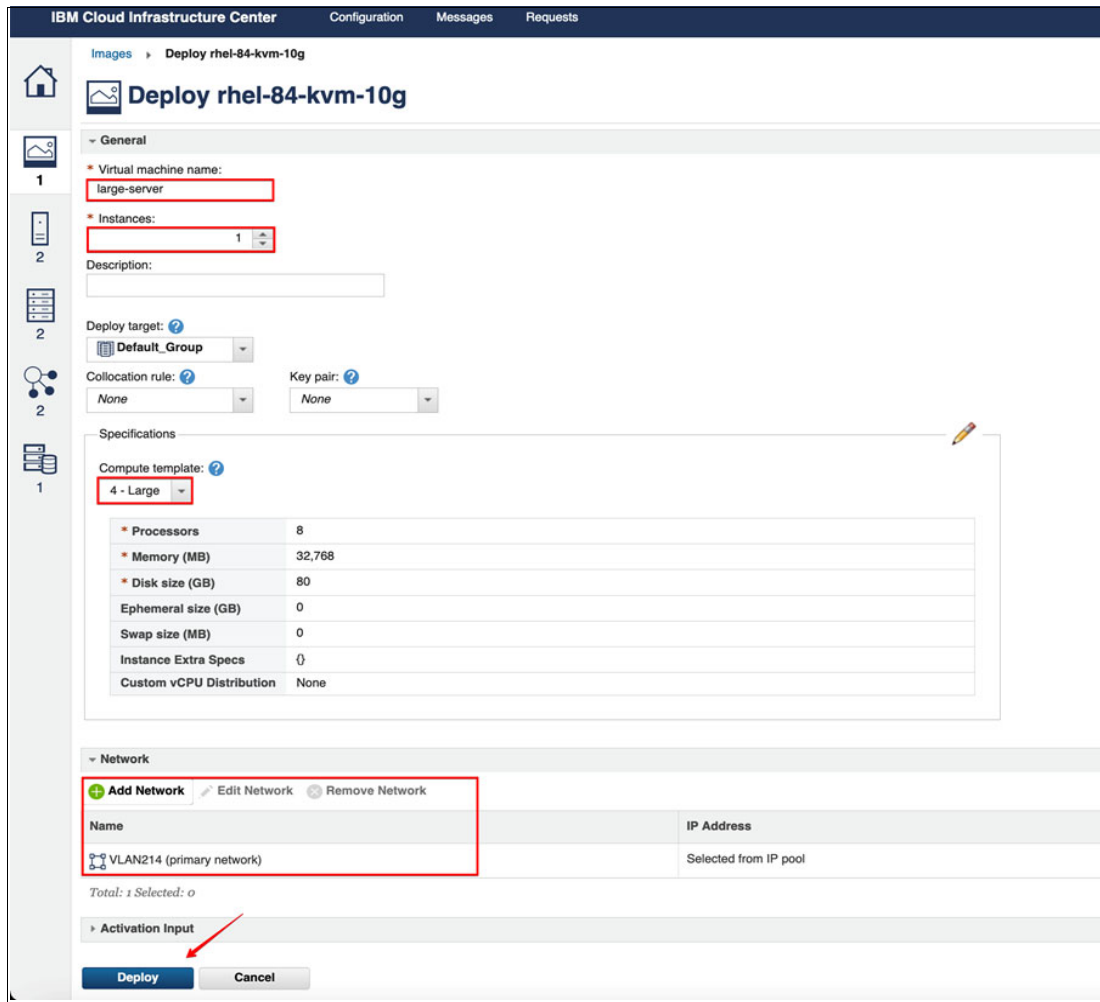


Figure 9-31 Deploying images

4. Click **Deploy**. The request is submitted to IBM Cloud Infrastructure Center. To monitor the new deployment, click the **Virtual Machines** page. The VM should be listed (see Figure 9-32).

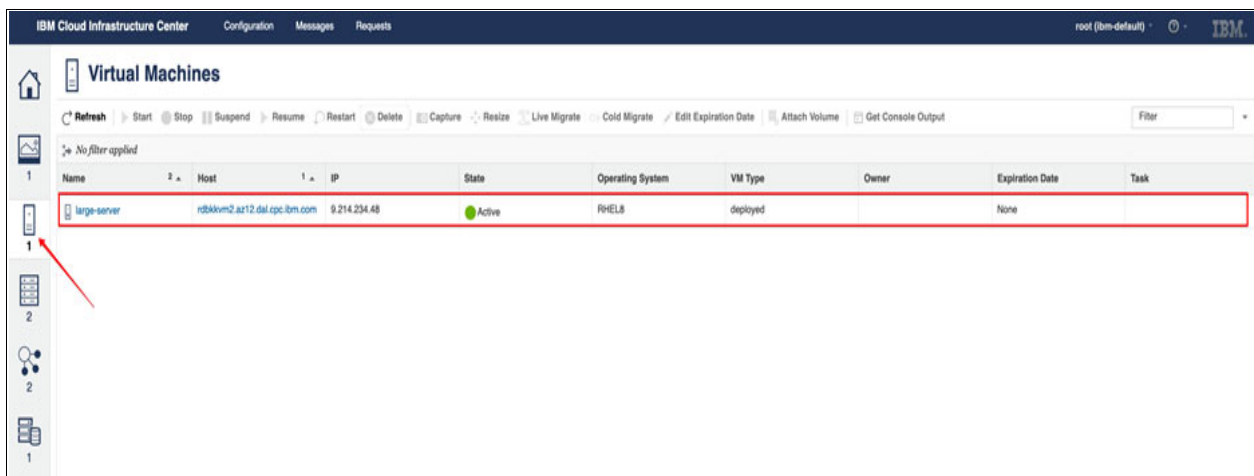


Figure 9-32 Virtual machines listed

- Optionally, you can acquire the IP address and attempt to connect to the server by using SSH keys.

Note: If you want to use SSH keys to connect to the new VMs, follow the steps that are described at this [IBM Documentation web page](#). Here, you also learn how to create your own SSH keys and add them into IBM Cloud Infrastructure Center.

Live migrating virtual machines

The IBM Cloud Infrastructure Center supports live migrating an instance from one compute node to another.

Complete the following steps to live migrate an instance on the UI:

- On the Virtual Machines page, select the VM that you want to use (see Figure 9-33).

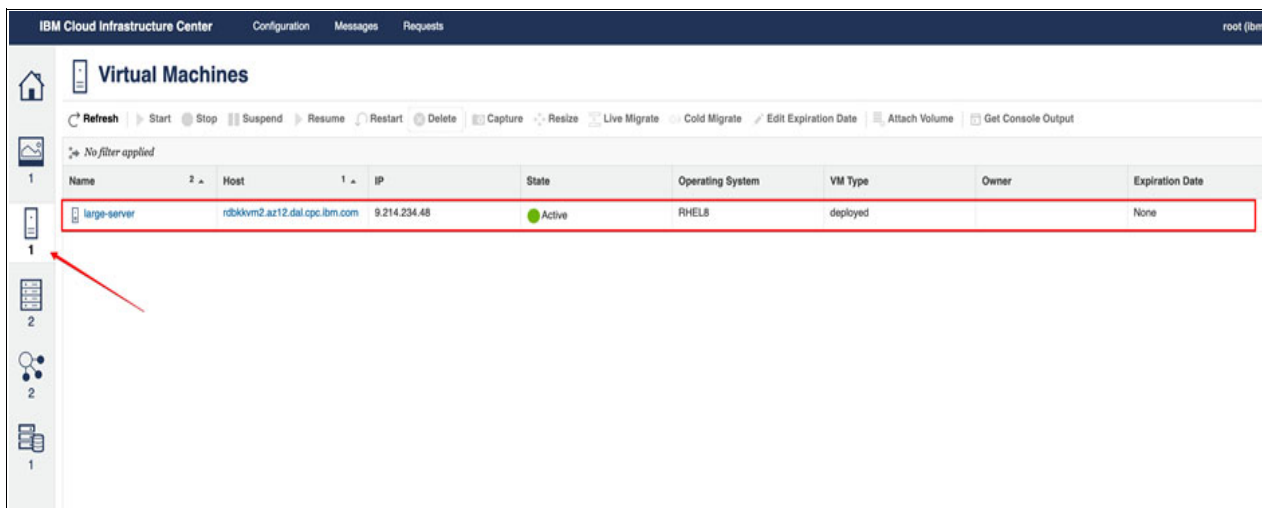


Figure 9-33 Selecting the VM

- Click **Live Migrate** at the top of the UI (see Figure 9-34).

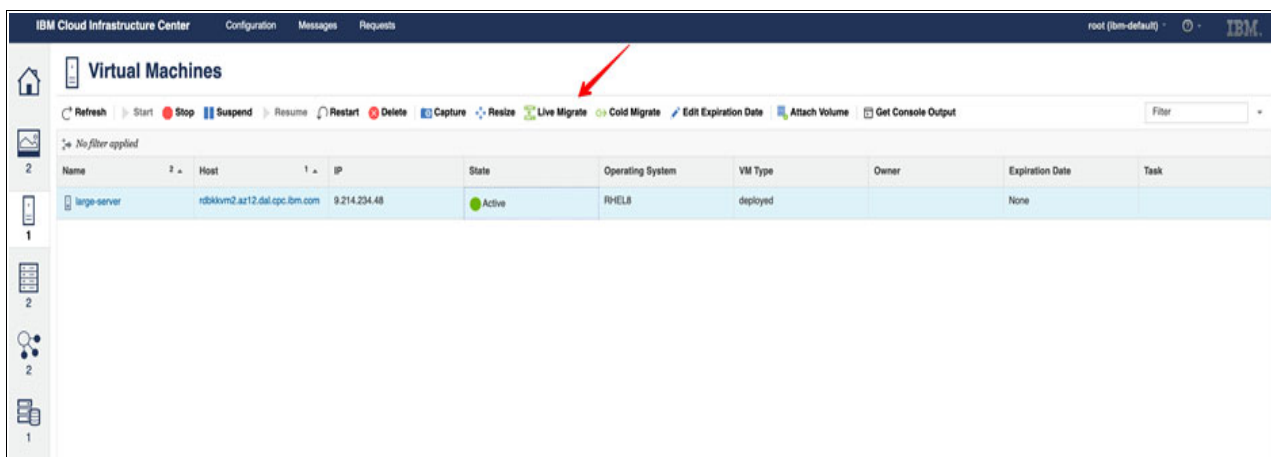


Figure 9-34 Virtual machines

- The destination hosts are listed in the Migrate dialog. Generally, two kinds of hosts are available: host group and host. Choose one of these types to migrate.

If you choose a host group, a host in the group is automatically chosen as the destination for this migration. If you choose a host, the host is the destination for this migration.

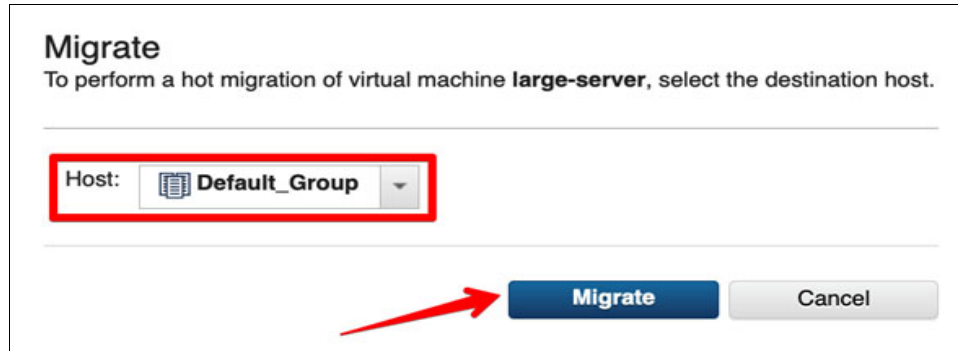


Figure 9-35 Migrating a VM

- The status of the VM appear as Migrating. The process can take several seconds to complete. When the process is complete, confirm whether the KVM LPAR name changed in the Host column (see Figure 9-36).



Figure 9-36 Migrating VM

Creating snapshots

Snapshots can be used to help prepare your environment for recovery. A snapshot allows for a complete copy of a VM at a point in time to be created. IBM Cloud Infrastructure Center provides the snapshot feature for the VMs.

Note: The VMs are created by using the RHCOS 4.2 or RHCOS 4.3 image do not support snapshots.

You can create a snapshot of the VM that was created or deployed in IBM Cloud Infrastructure Center. Assume that all the IBM Cloud Infrastructure Center services are available and one or more VMs were created on the host.

Complete the following steps to create the snapshot of the expected VM:

1. On the **Virtual Machines** page, select the VM that you want to use (see Figure 9-37).

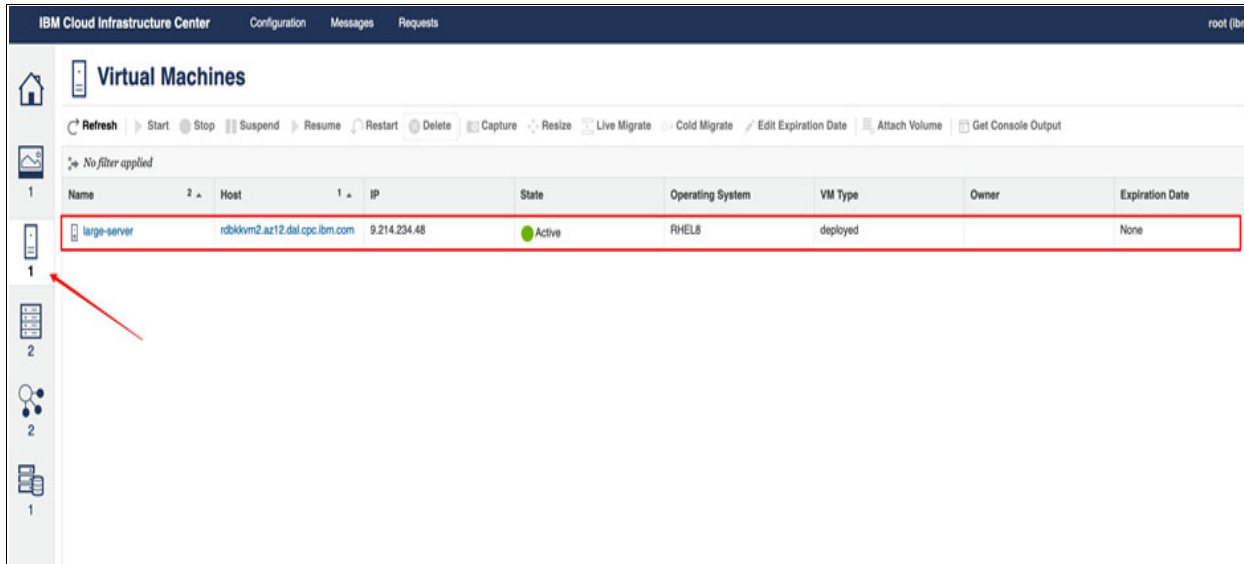


Figure 9-37 Virtual Machine migrated

2. Click **Capture**. In the pop-up window, enter the name of the capture result or do not enter any information to accept the default name.

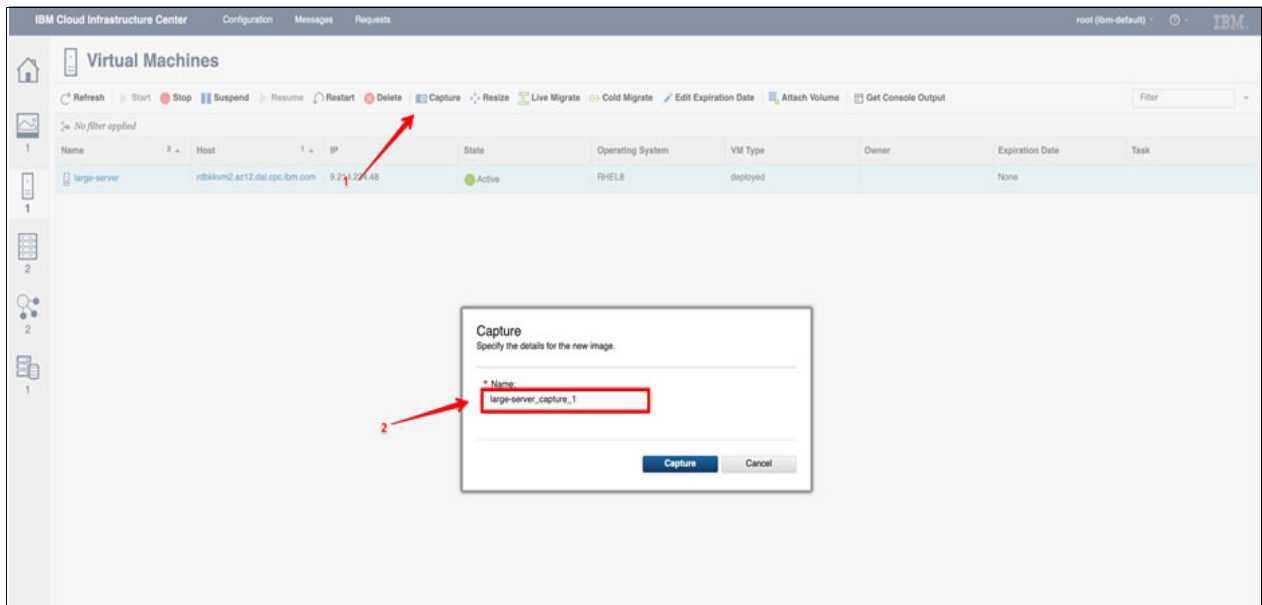


Figure 9-38 Snapshot creation

3. Click **Capture** to start the capture. The task state of the captured VM shows Capturing. The Capture state is displayed in the Images window. If the capture is successful, the final state turns to Active. Other information includes the name of the captured image, operating system, description, and the last updated time for the image.

In the Virtual Machines window, the state of the captured VM changes to Shutoff for the VMs that are deployed from local disks. For the VMs that are deployed from volumes, the state of the captured VM does not change (see Figure 9-39).

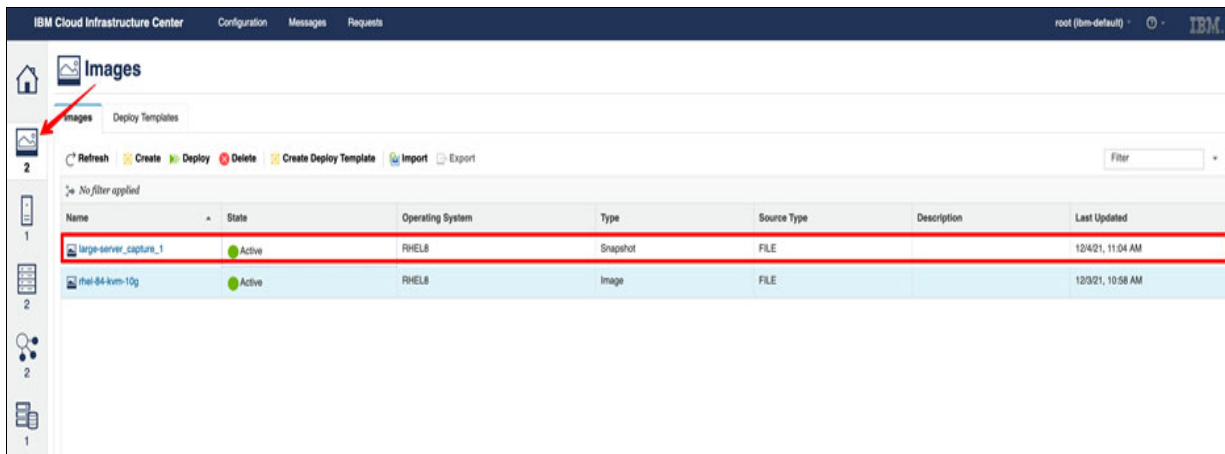


Figure 9-39 Image Snapshot display

- The captured image that is the snapshot can be used to deploy a new VM, similar to other uploaded images.

Note: The VM that is deployed from local disk is stopped during the capture, and the VM state remains Shutoff after it is captured. The state of the VM that is deployed from the remote volume remains Active after it is captured.

Resizing virtual machines

Resizing is the ability to change the type of a VM, which allows it to upscale or downscale according to a user's needs. You can resize a VM only when its status is in ACTIVE or SHUTOFF.

On the KVM hypervisor, the resize features the following behavior:

- VMs are resized to other KVM hosts or the same host on which they are originally located.
- The KVM resize is not a live resize. If VMs are active before resizing, they are stopped first and then captured. Then, they are deployed with the new type to the best fit candidate host. After the resizing finishes, the VMs are returned back to active state.
- For all memory, the CPU or disks of VMs can be changed, but the disks cannot be resized down.

Note: As of this writing, resizing on KVM is implemented by transferring the images between compute nodes over SSH. For KVM, you need host names to correctly resolve and passwordless the SSH access between your compute hosts.

Direct access from one compute host to another is needed to copy the VM file from one host to another. For more information, see this [IBM Documentation web page](#).

Complete the following steps to resize a VM. In this example, we resize a server from Large to Extra-Large (scale in):

1. In the Virtual Machines window, select the VM that you want to use (see Figure 9-40).

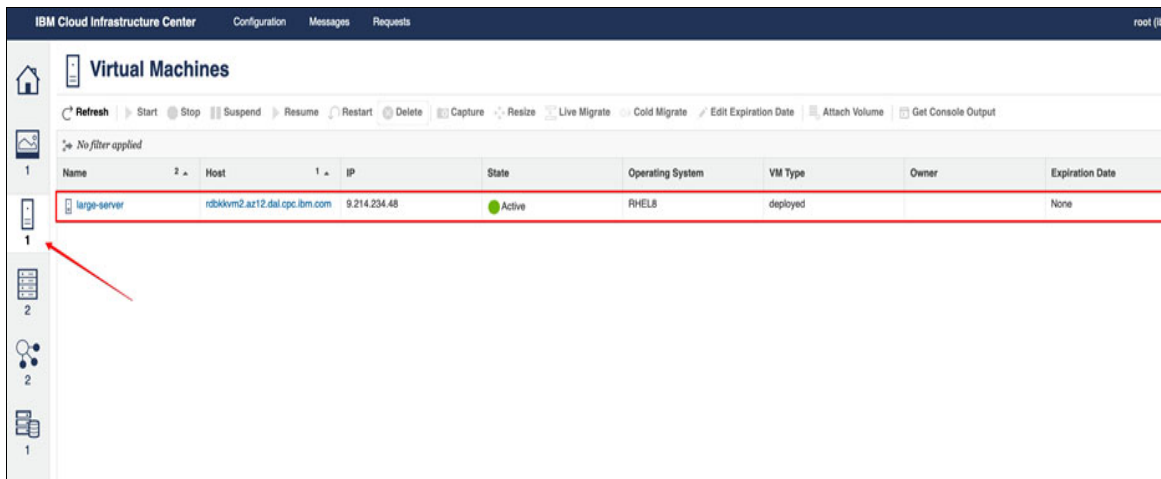


Figure 9-40 Resizing a VM

2. Click **Resize**. In the pop-up window, select the suitable Compute Template and then, click **Resize** (see Figure 9-41).

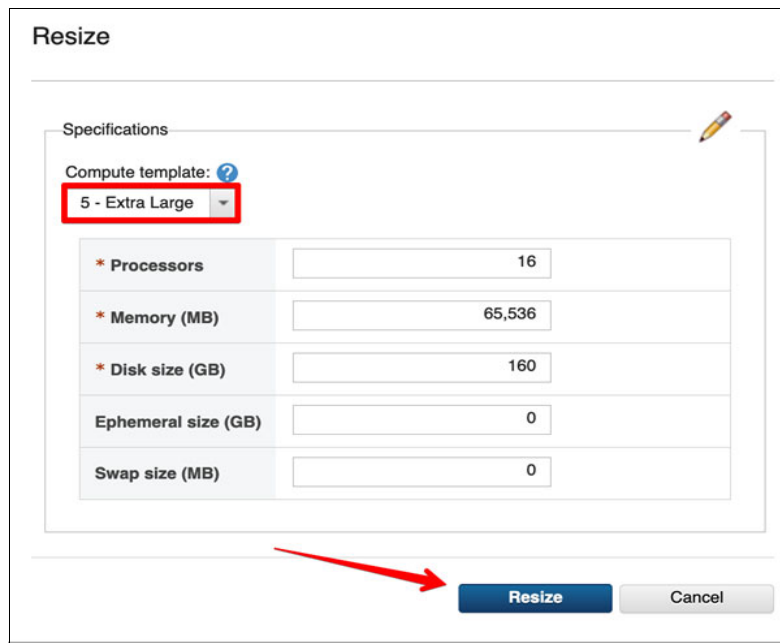


Figure 9-41 Resizing details

3. The status of the VM changes, as shown in Figure 9-42.

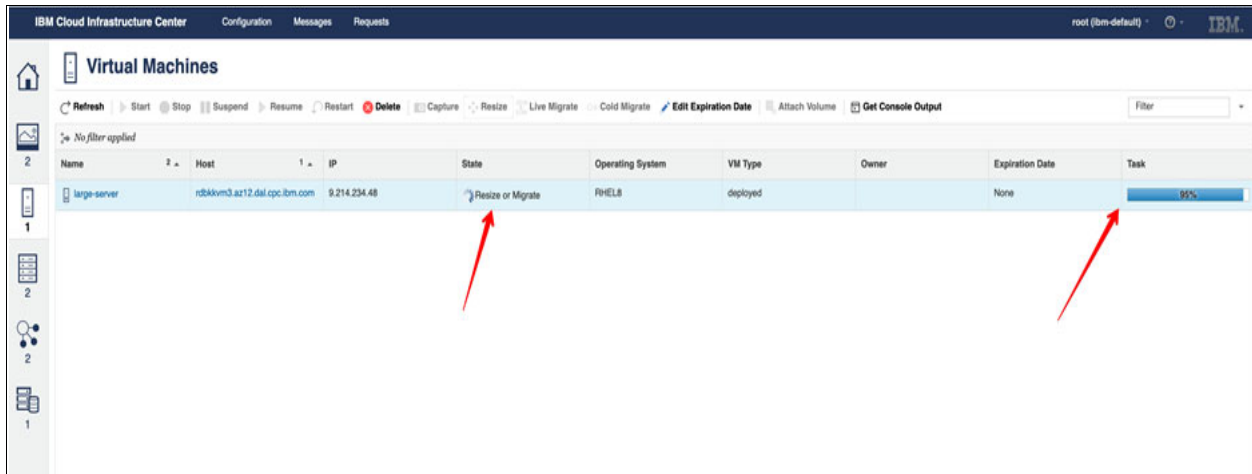


Figure 9-42 Resize in progress

4. After the resize process completes, click the server for more information, as shown in Figure 9-43. The extra server information is shown in Figure 9-44 on page 362.

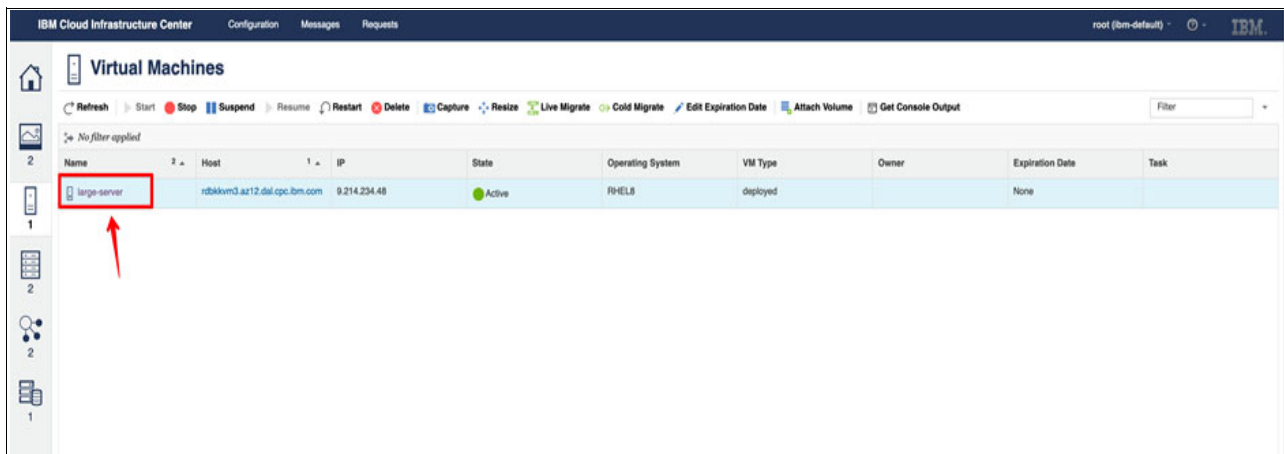


Figure 9-43 Resize complete

You see that the physical unit (PU), memory, and disk were changed to new values. The Host also might change because IBM Cloud Infrastructure Center redeploys the VM and another LPAR can be selected to accommodate the new size (see Figure 9-44).

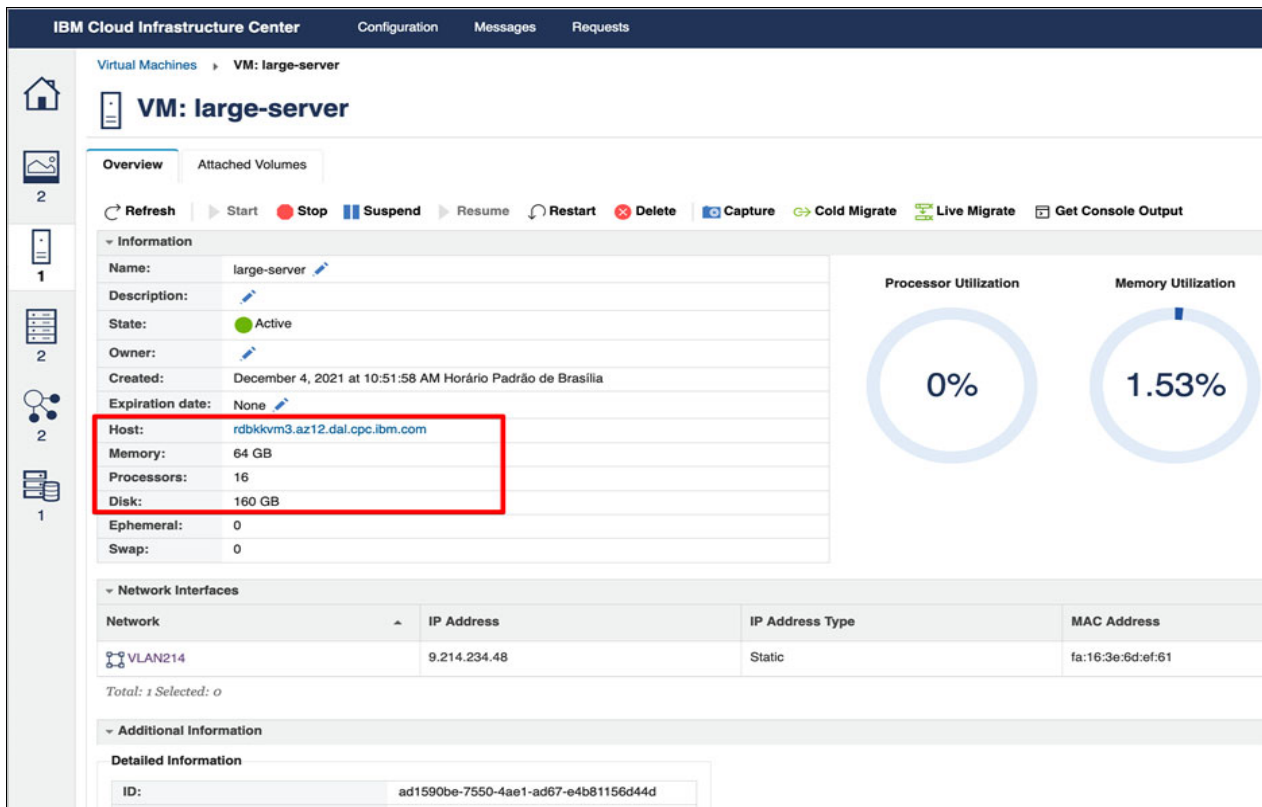


Figure 9-44 VM Larger Server

5. Complete the steps that are described in 9.2.4, “Extending the root file system” on page 352 to increase the disk and PV sizes.

Attaching volumes to virtual machines

Before you can attach volume to VM, ensure that the FCP devices are set up for the compute node.

To check whether the FCP devices exist on the compute node, run the `lszfcp` command, as shown in the following example:

```
[root@rdbkkvm2 ~]# lszfcp
0.0.f000 host0
0.0.f100 host1
0.0.f200 host2
0.0.f300 host3

[root@rdbkkvm3 ~]# lszfcp
0.0.f000 host0
0.0.f100 host1
0.0.f200 host2
0.0.f300 host3
```

Complete the following steps in the UI to attach volumes:

1. Open the Virtual Machines window.
2. Select the VM to which you want to attach volumes.
3. Click **Attach Volume** and you see a pop-up window that shows the volumes that you created.
4. Select a volume and click **Attach** to attach it to the VM.
5. Wait until the volume is in In-Use status.

Deleting volumes

Complete the following steps to delete volumes:

1. In the Data Volumes window of the Storage Providers window, select one or more volumes and click **Delete**.
2. Click **OK** in the confirm dialog.
3. Wait until the volumes are removed.

Note: Some types of volumes cannot be deleted, such as a volume that is in use, or the volume includes an associated snapshot. In this case, the Delete option is not available or error dialog is displayed after the Delete option is clicked.

Deleting virtual machines

Complete the following steps to remove a VM:

1. Select the **Virtual Machines icon** to open the Virtual Machines window (see Figure 9-45).

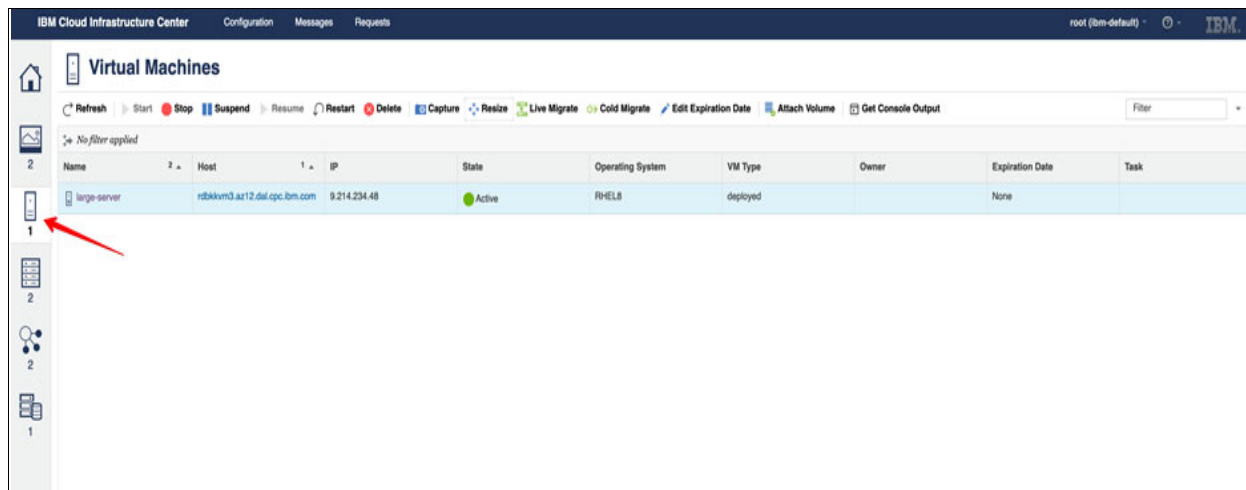


Figure 9-45 VM icon

2. Select the VM that is to be removed and then, click **Delete**. A pop-up window opens (see Figure 9-46).



Figure 9-46 Confirming the deletion

Note: The VM does not have SAN volumes. Therefore, IBM Cloud Infrastructure Center disables the volume deletion options.

3. Review your options and then, click **Delete** to remove the VM.

The VM disappears after few minutes if the process was successful.

9.3 Creating a bond for KVM administration network interfaces

This section describes commands to enable the two OSA cards that are used for administering KVMs.

As shown in Figure 9-47, we define b80 and b90 OSA cards into an active-backup bonding for high availability. The other two OSAs (b00 and b10) are used by the Linux VMs.

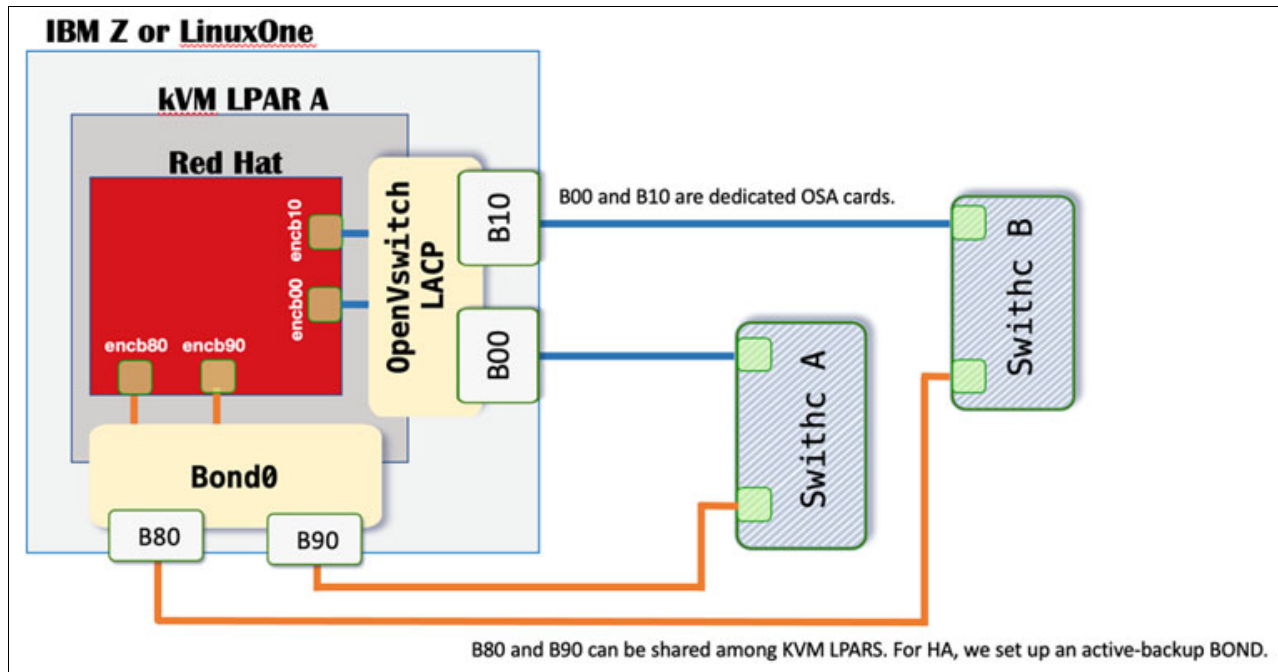


Figure 9-47 OSA bonding for high availability

This architectural decision was made so that the b80 and b90 OSA cards can be shared with any other KVM LPAR in the cluster and costs are reduced to avoid dedicating OSA cards to perform only KVM administration tasks.

The following commands were used to enable the network devices (b80, b90, and bond0) and set up the IP address on the bond0 interface:

Note: Use HMC to run these commands because the SSH connection is lost when the KVM administration interfaces are redefined.

- Define b80 and b90 OSA cards into an active-backup bonding for high availability:

```
cio_ignore -r b80-b82
cio_ignore -r b90-b92
znetconf -A
chzdev --enable qeth 0.0.0b80
chzdev --enable qeth 0.0.0b90
```

- ▶ Delete and redefine the network connections by using nmcli:


```
nmcli connection down 'System encb80'
nmcli connection delete 'System encb80'
nmcli connection down 'System encb90'
nmcli connection delete 'System encb90'
nmcli connection add type bond con-name bond0 ifname bond0 bond.options
"mode=active-backup,miimon=1000"
nmcli connection add type ethernet slave-type bond con-name bond0-port1 ifname
encb80 master bond0
nmcli connection add type ethernet slave-type bond con-name bond0-port2 ifname
encb90 master bond0
```
- ▶ Set the IP address, default GW and other IP settings:


```
nmcli connection modify bond0 ipv4.addresses '9.214.220.201/23'
nmcli connection modify bond0 ipv4.gateway '9.214.220.1'
nmcli connection modify bond0 ipv4.dns '9.0.128.50'
nmcli connection modify bond0 ipv4.dns-search 'az12.dal.cpc.ibm.com'
nmcli connection modify bond0 ipv4.dns-options 'single-request-reopen'
nmcli connection modify bond0 ipv4.method manual
```
- ▶ Verify the new bond connection:


```
nmcli connection up bond0
nmcli device
```
- ▶ Make auto-connection feature available for the slave interfaces:


```
nmcli connection modify bond0 connection.autoconnect-slaves 1
nmcli connection up bond0
```
- ▶ Confirm that bonding is up and operational:


```
cat /proc/net/bonding/bond0
```
- ▶ Make the auto-connection feature available for slave interfaces:


```
nmcli connection modify bond0 connection.autoconnect-slaves 1
```
- ▶ Confirm that bonding is up and operational:


```
cat /proc/net/bonding/bond0
```

You should receive the following output if bonding is operational:

```
Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)

Bonding Mode: fault-tolerance (active-backup)
Primary Slave: None
Currently Active Slave: encb80
MII Status: up
MII Polling Interval (ms): 1000
Up Delay (ms): 0
Down Delay (ms): 0
```

Peer Notification Delay (ms): 0

Slave Interface: encb80

MII Status: up

Speed: 25000 Mbps

Duplex: full

Link Failure Count: 0

Permanent HW addr: 96:14:39:0a:c3:64

Slave queue ID: 0

Slave Interface: encb90

MII Status: up

Speed: 25000 Mbps

Duplex: full

Link Failure Count: 0

Permanent HW addr: 9a:2b:41:c1:6b:95

Slave queue ID: 0



A

Live Virtual Server Migration

This appendix describes a solution with which customers that are running Linux on Kernel-based Virtual Machine (KVM) can use the Live Migrate function that is offered in KVM hypervisor to relocate Linux guests around the KVM cluster.

This appendix includes the following topics:

- ▶ “Introduction” on page 370
- ▶ “Live migration phases” on page 370
- ▶ “Provisioning two servers” on page 371
- ▶ “Performing a live migration” on page 375

Introduction

A KVM cluster consists of KVM Linux systems in which all members can access shared resources, including DASD volumes, Ethernet LAN segments, and storage area networks (SANs). System management becomes easier because the members can be serviced, managed, and administered as though they are one integrated system.

KVM clustering helps to meet horizontal growth of Z workload. With Live Migrate feature, Linux virtual servers (guest virtual machines [VMs]) can be relocated to another member in the cluster.

That way, organizations can roll out hardware and software maintenance and upgrades without disrupting the business. This capability makes a significant contribution to continuous availability.

To achieve higher levels of system, application, and data availability, organizations can consider transitioning from stand-alone KVM systems to a KVM cluster.

You also can consider implementing Linux Health Checker to identify issues before they cause problems. It helps to detect configuration errors, deviations from best practices, single point-of-failures, unused accelerator hardware, hardware that is running in degraded mode, and so on.

To migrate a running virtual server between hosts without affecting the virtual servers, the source and destination hosts must be connected and access the same or equivalent systems resources, storage devices, and networks.

For more information about live migration, see this [IBM Documentation web page](#).

Live migration phases

The migration of a virtual server from a source to a destination host consists of two phases: the live phase and the stopped phase.

These phases are described next.

Live phase

While the virtual server is running, its memory pages are transferred to the destination host. During the live phase, the virtual server might continue to modify memory pages. These pages are called *dirty pages*, which must be retransmitted.

QEMU continuously estimates the time that it needs to complete the migration during the stopped phase. If this estimated time is less than the specified maximum downtime for the virtual server, the virtual server enters the stopped phase of the migration.

If the virtual server changes memory pages faster than the host can transfer them to the destination, the migration command option `--auto-converge` can be used to throttle down the CPU time of the virtual server until the estimated downtime is less than the specified maximum downtime. If you do not specify this option, the virtual server might never enter the stopped phase because too many dirty pages must be migrated.

This mechanism works for average virtual server workloads. Workloads that are memory intensive might require the extra specification of the `--timeout` option. This option suspends the virtual server after a specified amount of time and avoids the situation where throttling down the CPU cannot catch up with the memory activity and thus, in the worst case, the migration operation never stops.

Stopped phase

During the stopped phase, the virtual server is paused. The host uses this downtime to transfer the rest of the dirty pages and the virtual server's system image to the destination.

If the virtual server makes use of storage keys, they are also migrated during this phase.

Provisioning two servers

This section describes how to deploy two Red Hat Linux VMs by using Red Hat Kickstart.

Before starting, the following prerequisites must be met:

- ▶ Red Hat Enterprise Linux DVD1 is available
- ▶ Two hostname names and two IP addresses are available for the subnet where the VMs are to be deployed. In our case, we chose 9.214.232.208 for `kvm-vm-001` and 9.214.232.209 for `kvm-vm-002` servers.

The steps that are listed in this section provide a quick method to create a Red Hat 8.4 VM by using Red Hat Kickstart process.

For more information about Red Hat Kickstart, see this [Red Hat Documentation web page](#).

Complete the following steps on only the first KVM compute node (`rdbkvm2`). It deploys a new VM:

1. Run the following command to create an encrypted password, which is used later to update the Red Hat Kickstart file:

```
python -c 'import crypt,getpass;pw=getpass.getpass();print(crypt.crypt(pw) if (pw==getpass.getpass("Confirm: ")) else exit()'
```

Note: When prompted, enter the initial password that is to be used for the root ID. Make a note of this encrypted password because it is need later.

2. Create the `/gpfs/img` folder:

```
mkdir -pv /gpfs/img
```

3. Upload the Red Hat DVD image. (If you do not have the image, download it from this [Red Hat web page](#)):

```
ls -la /gpfs/img/rhel-8.4-s390x-dvd.iso
-rw-r--r--. 1 root root 3783262208 Dec  3 11:14
/gpfs/img/rhel-8.4-s390x-dvd.iso
```

4. Create the `/gpfs/rhel-8.4-s390x-VM.cfg` file by using the content that is shown in Example A-1. You must update the settings that are highlighted in **bold** with the encrypted root password that was created in Step 1 and update the IP address of the new virtual server.

Example A-1 Content for /gpfs/rhel-8.4-s390x-VM.cfg file

```
text
lang en_US.UTF-8
keyboard us
rootpw
$6$y9kVIF0tWVChn3J0$qPP9arf1Ah1VRB4BJCbXALMYrKGJGSzD.6xGfUSnFAmB9cyUbrD7e4d56y
T.L1fC52ptHatKMxrmhT20x/eM1 --iscrypted
timezone America/New_York
poweroff
zerombr
ignoredisk --only-use=vda
clearpart --all
# Static IP address with gateway to allow network communication - Update this
section
network --hostname=kvm-vm-001 --bootproto=static --ip=9.214.232.208
--netmask=255.255.255.0 --gateway=9.214.232.1 --nameserver=9.0.128.50

part /boot --fstype=xfs --size=1024 --asprimary

part pv.0 --fstype=lvmPV --size=8192 --grow
volgroup systemvg --pesize=8192 pv.0
logvol / --vgname=systemvg --name=root --fstype=xfs --size=1024
logvol /usr --vgname=systemvg --name=usr --fstype=xfs --size=4096
logvol /home --vgname=systemvg --name=home --fstype=xfs --size=128
logvol /opt --vgname=systemvg --name=opt --fstype=xfs --size=128
logvol /tmp --vgname=systemvg --name=tmp --fstype=xfs --size=128
logvol /var --vgname=systemvg --name=var --fstype=xfs --size=1024
logvol swap --vgname=systemvg --name=swap --fstype=swap --size=1024

syspurpose --role="Red Hat Enterprise Linux Server" --sla="Standard"
--usage="Development/Test"
auth --passalgo=sha512
selinux --permissive
firewall --disabled
skipx
firstboot --disable

%packages
@^server-product-environment
kexec-tools
bind-utils
device-mapper-multipath
e2fsprogs
git
iproute
ksh
net-tools
cloud-init
cloud-utils-growpart
procps
```

```

unzip
vim
zip
file
perl-Net-Ping
binutils
wget
rsync
postfix
gettext
yum
%end

%post
subscription-manager unregister
touch /etc/sudo.env
cat /dev/null > /etc/multipath/bindings
cat /dev/null > /etc/multipath/wwids
cat /usr/share/doc/device-mapper-multipath-0.4.9/multipath.conf >
/etc/multipath.conf

sed -i -e '/^users:$/,/^ - default$/d' -e 's/^disable_root: 1/disable_root: 0/'
-e 's/^ssh_pwauth: 0/ssh_pwauth: 1/' /etc/cloud/cloud.cfg

echo "Installation completed."

%end

```

5. Export BASH variables to help create the VM creation. Update MachineName for the name of the VM that you are deploying now.

```

export ISO="/gpfs/img/rhel-8.4-s390x-dvd.iso"
export MachineName='kvm-vm-001'

```

Consider the following points:

- ISO contains the path for the Red Hat 8.4 DVD .iso file, which must be downloaded.
- MachineName contains the name of the VM that is created in IBM Cloud Infrastructure Center.

6. Ensure that the virt-install package is installed:

```

yum install -y virt-install

```

7. Create the QCOW2 file:

```

qemu-img create -f qcow2 -o preallocation=metadata /gpfs/${MachineName}.qcow2
10G

```

8. Use the following command to automatically create the Red Hat VM:

```

virt-install --virt-type=kvm --name=${MachineName} --vcpus=2 --memory=4096
--location="${ISO}" --disk path=/gpfs/${MachineName}.qcow2,format=qcow2
--network default --noreboot --initrd-inject=/gpfs/rhel-8.4-s390x-VM.cfg
--extra-args "inst.ks=file:/rhel-8.4-s390x-VM.cfg"

```

Note: The installation takes some minutes to complete. Wait until the process is completed, at which time you receive the following output that is shown in Figure A-1.

```
Verifying vim-filesystem.noarch (672/676)
Verifying volume_key-libs.s390x (673/676)
Verifying wget.s390x (674/676)
Verifying xdg-utils.noarch (675/676)
Verifying xkeyboard-config.noarch (676/676)
.
Installing boot loader
..
Performing post-installation setup tasks
.
[terminated]
```




Figure A-1 Progress output

9. Repeat the steps that are shown in Step 8 for the second VM. Remember to update the content that is highlighted in **bold** in Example A-1 on page 372 with the `kvm-vm-002` name and IP address.
10. Now that `kvm-vm-001` and `kvm-vm-002` are deployed, start the servers:

```
# virsh list --all
Id   Name           State
-----
-    kvm-vm-001    shut off
-    kvm-vm-002    shut off
# virsh start kvm-vm-001
Domain kvm-vm-001 started
# virsh start kvm-vm-002
Domain kvm-vm-002 started
# virsh list --all
Id   Name           State
-----
18   kvm-vm-001    running
19   kvm-vm-002    running
```

If problems occur in building the servers, run the following commands to shut down and remove the definition of the servers before attempting the installation again:

```
virsh shutdown kvm-vm-001
virsh undefine kvm-vm-001
```

Performing a live migration

The commands that are described in this section are useful in the context of a live migration.

Procedure

Complete the following steps to perform a live migration:

1. Verify the default tolerable downtime for the virtual servers:

```
# virsh migrate-getmaxdowntime kvm-vm-001
300
# virsh migrate-getmaxdowntime kvm-vm-002
300
```

Optional: You can specify a tolerable downtime for a virtual server during a migration operation by using the virsh **migrate-setmaxdowntime** command. The specified value is used to estimate the point in time when to enter the stopped phase.

2. Use the `maxdowntime` value if problems are encountered while migrating a server. This value might be too low for the migration to complete.

You can still issue the following command during the migration process:

```
# virsh migrate-setmaxdowntime <VS> <milliseconds>
```

Optional: You might want to limit the bandwidth that is provided for a migration.

To set or modify the maximum bandwidth, use the virsh **migrate-setspeed** command (see this [IBM Documentation web page](#)):

```
# virsh migrate-setspeed <VS> --bandwidth <mebibyte-per-second>
```

You can display the maximum bandwidth that is used during a migration by using the virsh **migrate-getspeed** command (see this [IBM Documentation web page](#)):

```
# virsh migrate-getspeed <VS>
```

3. Start a live migration of a virtual server from `rdbkkvm2` to `rdbkkvm3` by using the virsh **migrate** command with the `--live` option:

```
# virsh --keepalive-interval 10 migrate --live --persistent --undefinesource
--timeout 1200 --verbose kvm-vm-001 qemu+ssh://rdbkkvm3/system
Migration: [100 %]
```

The command uses the following syntax:

```
# virsh migrate --live <command-options> <VS>
qemu+ssh://<destination-host>/system
```

Note: When virsh connects to the destination host by way of SSH, you are prompted to enter a password. For more information about avoiding the need to enter a password, see [this web page](#).

Where:

- `<command-options>` are the virsh **migrate** command options.
- `<destination-host>` is the name of the destination host.

- `<mebibyte-per-second>` is the migration bandwidth limit in MiBps.
- `<milliseconds>` is the number of milliseconds that is used to estimate the point in time when the virtual server enters the stopped phase.
- `<VS>` is the name of the virtual server as specified in its domain configuration-XML file.

Consider the following points:

- ▶ Optional: The use of the `--auto-converge` and the `--timeout` options ensure that the migration operation completes.
- ▶ Optional: To avoid connectivity loss during a time-consuming migration process, increase the `virsh keepalive` interval (see [this web page](#)):

```
# virsh --keepalive-interval <interval-in-seconds>
```

- ▶ The use of the `virsh --keepalive-interval` and `--keepalive-count` options preserves the communication connection between the host that initiates the migration and the `libvirtd` service on the source host during time-consuming processes.

- ▶ Use the `keepalive` options if:

- The virtual server is running a memory-intensive workload so that it might need to be suspended to complete the migration.
- You can use an increased timeout interval. The following default values are used:

- `keepalive interval`: 5 seconds
- `keepalive count`: 6

These defaults can be changed in `/etc/libvirt/libvirtd.conf`, as shown in the following example:

```
# virsh --keepalive-interval 10 migrate --live --persistent
--undefinesource \
--timeout 1200 --verbose vserv1 qemu+ssh://kvmhost/system
```

This example shows an increase of 10 seconds in the `keepalive` interval of the connection to the host.

- ▶ Optional: Perform disk migration for any virtual block devices that are backed by local resources on the source host. Such local host resource can be, for example, image files in the host file system or PCIe-attached NVMe devices.

Specify the `--copy-storage-all` or `--copy-storage-inc` option with the `--migrate-disks` option to copy image files or file systems on NVMe devices to the destination host.

Restriction

Disk migration is possible for writable virtual disks only.

Example 1

One example of a read-only disk is a virtual DVD. If in doubt, check your domain configuration-XML. If the disk device attribute of a disk element is configured as `cdrom`, or contains a read-only element, the disk cannot be migrated.

Consider the following example in which the `qcow2` image `/var/libvirt/images/vdd.qcow2` is copied to the destination host (assuming that the VDD is configured as shown here:

```
<disk type="file" device="disk">
  <driver name="qemu" type="qcow2" io="native" cache="none"/>
  <source file="/var/lib/libvirt/images/vdd.qcow2"/>
```



```
<target dev="vdd" bus="virtio"/>
<address type="ccw" cssid="0xfe" ssid="0x0" devno="0x0004"/>
</disk>
```

First, a qcow2 image is created on the destination host:

```
[root@destination]# qemu-img create -f qcow2 \
/var/lib/libvirt/images/vdd.qcow2 1G
```

Then, the virsh **migrate** command is run on the source host:

```
[root@source]# virsh migrate --live
--copy-storage-all --migrate-disks vdd \
vserv2 qemu+ssh://zhost/system
```

Results

The virtual server is not destroyed on the source host until it is migrated to the destination host.

If an error occurs during migration, the resources on the destination host are cleaned up and the virtual server continues to run on the source host.

Example 2

In this example, a live migration of the virtual server vserv3 to the destination host zhost is started.

The virtual server is transient on zhost; that is, after vserv3 is stopped on zhost, its definition is deleted. After a successful migration, the virtual server is destroyed on the source host, but still is defined.

If the migration operation is not ended within 300 seconds, the virtual server is suspended while the migration continues.:

```
# virsh migrate --live --auto-converge --timeout 300 vserv3
qemu+ssh://zhost/system
```

Next steps

You can verify whether the migration completed successfully by looking for a running status of the virtual server on the destination; for example, by using the virsh **list** command:

```
# virsh list
Id   Name           State
-----
 5   kvm-vm-001    running
```

You also can cancel an on-going migration operation by using the virsh **domjobabort** command:

```
# virsh domjobabort <VS>
```




B

Kernel-based Virtual Machine LPAR live migration

This appendix describes an example of a virtual machine (VM) live migration between two different logical partitions (LPARs) in the same physical machine.

Overview

Figure B-1 shows an overview of the environment that was used for our example.

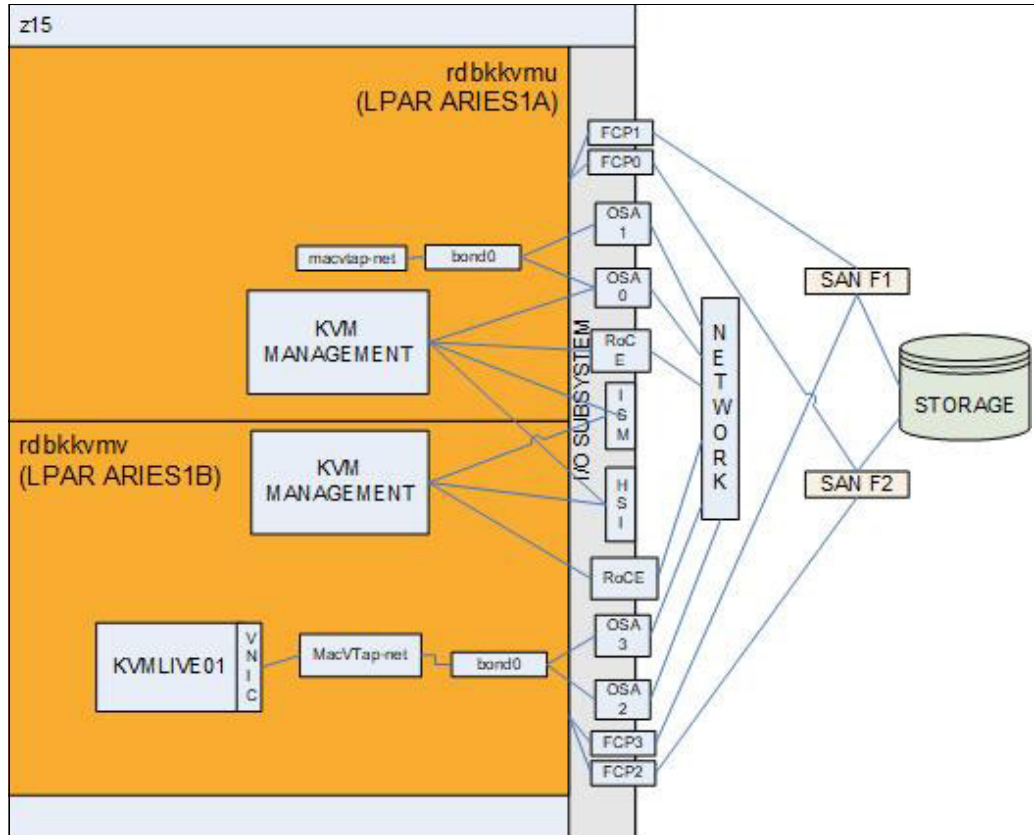


Figure B-1 Live migration physical resources

Before you attempt this example, see 2.2.7, “Linux virtual machine live migration” on page 36 for more information about planning.

For more information about live migration, see [Linux on Z and LinuxONE: KVM Virtual Server Management](#).

In this example, we move the `kvm live01` VM guest from the KVM host, `rdbkkvmv`, to `rdbkkvmu` by using the live migration feature.

The following information was collected for this example:

- ▶ `rdbkkvmv` KVM host:
 - Management IP address: 9.76.61.179
 - HiperSockets IP address: 100.150.233.43
 - MacVTap network: `macvtap-net`
 - OSA IP address with the same physical network (PNET) ID as ISM interface: 129.40.23.242
 - FCP device adapters: B90B and C90B

- ▶ rdbkvmu KVM host:
 - Management IP address: 9.76.61.184
 - HiperSockets IP address: 100.150.233.42
 - MacVTap network: macvtap-net
 - OSA IP address with the same PNET ID as ISM interface: 129.40.23.243
 - FCP device adapters: B90A and C90A
- ▶ kvmlive01 VM guest:
 - vNIC IP address: 9.76.61.40
 - LUN used by the operating system: 4001400F00000000

Considerations for our environment

On IBM Z, it is possible to proceed with live migration on all available interfaces: SMC-R (RoCE), OSA, SMC-D (ISM), or HiperSocket.

In this lab, we perform the live migration between two different LPARs in the same Z platform by using ISM (SMC-D) devices to communicate.

The VM network interface, macvtap-net, must have the same name in both LPARs. Example B-1 shows the (domain) network definition of the VM guest.

Example B-1 kvmlive01 network definition

```
<interface type='direct'>
  <mac address='52:54:00:08:cd:b9' />
  <source network='macvtap-net' dev='bond0' mode='bridge' />
  <target dev='macvtap1' />
  <model type='virtio' />
  <alias name='net0' />
  <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0001' />
</interface>
```

The VM guest uses a block device as a disk source with the same dm-uuid in both LPARs. Example B-2 shows the (domain) disk definition of the kvmlive01 VM guest.

Example B-2 kvmlive01 disk definition

```
<disk type='block' device='disk'>
  <driver name='qemu' type='raw' cache='none' io='native' />
  <source
dev='/dev/disk/by-id/dm-uuid-mpath-36005076309ffd145000000000000010f' />
  <backingStore />
  <target dev='vda' bus='virtio' />
  <alias name='virtio-disk0' />
  <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0000' />
</disk>
```

The source and destination KVM requires access to the VM guest disks. Example B-3 shows the disk device is present on both machines.

Example B-3 Verifying device access

```
root@rdbkkvmu:/dev/disk/by-id# ls | grep 10f
dm-uuid-mpath-36005076309ffd145000000000000010f
root@rdbkkvmv:/dev/disk/by-id# ls | grep 10f
dm-uuid-mpath-36005076309ffd145000000000000010f
```

Performing live migration

This example shows how to perform live migration by using the SMC-D feature. For more information about how to enable an SMC-D environment between KVM hosts, see “Enabling SMC-D” on page 70.

For this example, the SSHD service on rdbkkvmu was enabled also to listen on the ISM (SMC-D) interface by using similar steps that are described in “Enabling SMC-D” on page 70. Example B-4 shows the command that is used to perform a live migration from rdbkkvmv to rdbkkvmu.

Example B-4 Live migration command

```
root@rdbkkvmv:/home/lxadmin# virsh migrate kvmlive01
qemu+ssh://129.40.23.243/system
```

Example B-5 shows the SMC-D feature use during the live migration process.

Example B-5 Live migration by using SMC-D

```
root@rdbkkvmu:/home/lxadmin/smc-tools/smc-tools-1.2.0# ./smcss -a
State      UID   Inode  Local Address      Peer Address      Intf
Mode
INIT       00000 0000000
ACTIVE     00000 0462309 129.40.23.243:22   129.40.23.242:40804 0000
SMCD
INIT       00000 0000000
LISTEN     00000 0405985 0.0.0.0:22
```

Example B-6 shows the `kvmlive01` migrated between two different KVM hosts. It reflects being shut off in the previous host (`rdbkkmv`) and running in the new one (`rdbkkmv`).

Example B-6 Live migration verification

```
root@rdbkkmv:/home/lnxadmin/smc-tools/smc-tools-1.2.0# virsh list --all
 Id   Name           State
-----
 12   kvmlive01      running
root@rdbkkmv:/home/lnxadmin/smc-tools/smc-tools-1.2.0# virsh domstate kvmlive01
--reason
running (migrated)

root@rdbkkmv:/home/lnxadmin# virsh list --all
 Id   Name           State
-----
  3   RHEL77         running
 12   instance-0000010 running
 -   kvmlive01      shut off
root@rdbkkmv:/home/lnxadmin# virsh domstate kvmlive01 --reason
shut off (migrated)
```

Example B-7 shows the connectivity during the live migration.

Example B-7 Connectivity test during live migration

```
C:\Users\user10>ping 9.76.61.40 -t

Pinging 9.76.61.40 with 32 bytes of data:
Reply from 9.76.61.40: bytes=32 time=18ms TTL=48
Reply from 9.76.61.40: bytes=32 time=18ms TTL=48
Reply from 9.76.61.40: bytes=32 time=18ms TTL=48
Request timed out.
Reply from 9.76.61.40: bytes=32 time=27ms TTL=48
Reply from 9.76.61.40: bytes=32 time=18ms TTL=48
```



Scripts for SLES guest installation

This appendix describes the scripts that help you to simplify the set up for an AutoYAST installation and includes the following topics:

- ▶ “Preparing and setting up for AutoYAST installation” on page 386
- ▶ “AutoYAST configuration file for KVM guest” on page 387

Preparing and setting up for AutoYAST installation

Use script that is shown in Example C-1 to install a SLES KVM virtual machine.

Example C-1 prep-for-install.sh

```
#!/bin/bash

# KVM Host variables
AUTOYAST_FILE=/root/autoyast.xml
ISO_FILE=/var/lib/libvirt/images/isos/SLE-12-SP5-Server-DVD-s390x-GM-DVD1.iso
INSTALL_DIR=/var/lib/libvirt/images/sles12sp5-install

# oracle19c VM variables
VM_OS_DISK=/dev/disk/by-path/ccw-0.0.fa00-fc-0x500507630703d3b3-lun-0x4001403d00000000
VM_ORACLE_DISK=/dev/disk/by-path/ccw-0.0.fa00-fc-0x500507630703d3b3-lun-0x4001403e00000000
VM_ORACLEDB_DISK=/dev/disk/by-path/ccw-0.0.fa01-fc-0x500507630703d3b3-lun-0x4001403f00000000
0
VEPA_NIC=eth1
VEPA_NIC_DEVNO=0x0810

if test ! -e ${ISO_FILE}; then
    echo
    echo "Please copy $(basename ${ISO_FILE}) to $(dirname ${ISO_FILE})."
    echo "Quitting..."
    exit 1
fi
if test ! -e ${AUTOYAST_FILE}; then
    echo
    echo "Please copy $(basename ${AUTOYAST_FILE}) to $(dirname ${AUTOYAST_FILE})."
    echo "Quitting..."
    exit 1
fi

echo "Setting up environment for autoyast installation."
mkdir -p ${INSTALL_DIR}
TMP_MNT=$(mktemp -d)
mount ${ISO_FILE} ${TMP_MNT} > /dev/null 2>&1
cp ${TMP_MNT}/boot/s390x/linux ${INSTALL_DIR}
cp ${TMP_MNT}/boot/s390x/initrd ${INSTALL_DIR}
umount ${TMP_MNT}
rmdir ${TMP_MNT}
mkisofs -quiet ${AUTOYAST_FILE} > ${INSTALL_DIR}/autoyast.iso

echo "Generating oracle19c virtual machine xml file for SLES installation."
for disk in ${VM_OS_DISK} ${VM_ORACLE_DISK} ${VM_ORACLEDB_DISK}; do
    if test ! -e ${disk}; then
        echo
        echo "${disk} is not defined."
        echo "Quitting..."
        exit 1
    fi
done
if test $(lsqeth | grep -c ${VEPA_NIC}) -eq 0; then
    echo
    echo "${VEPA_NIC} is not defined."
    echo "Quitting..."
    exit 1
fi
virt-install --name oracle19c --vcpus 4 --memory 4096 \
```

```

--disk path=${VM_OS_DISK},cache=none,io=native,target=vda \
--disk path=${VM_ORACLE_DISK},cache=none,io=native,target=vdb \
--disk path=${VM_ORACLEDB_DISK},cache=none,io=native,target=vdc \
--network
type=direct,source=${VEPA_NIC},address.type=ccw,address.cssid=0xfe,address.ssid=0x0,address
.devno=${VEPA_NIC_DEVNO} \
--cdrom ${ISO_FILE} \
--disk path=${INSTALL_DIR}/autoyast.iso,device=cdrom \
--boot kernel=${INSTALL_DIR}/linux,initrd=${INSTALL_DIR}/initrd,\
kernel_args="self_update=0 install=cd:/ autoyast=device://sr1/autoyast.xml" \
--print-xml 1 > /root/ora_install.xml

echo "Defining oracle19c virtual machine for runtime administration."
virsh -q define /root/ora_install.xml
virt-xml -q oracle19c --remove-device --disk path=${INSTALL_DIR}/autoyast.iso,device=cdrom
virt-xml -q oracle19c --edit --boot kernel=,initrd=,kernel_args=
virt-xml -q oracle19c --edit --events on_reboot=restart

exit 0

```

AutoYAST configuration file for KVM guest

Example C-2 shows the autoyast.xml file that is used to automatically install SLES 12 SP5.

Example C-2 autoyast.xml

```

<?xml version="1.0"?>
<!DOCTYPE profile>
<profile xmlns="http://www.suse.com/1.0/yast2ns"
xmlns:config="http://www.suse.com/1.0/configs">
  <add-on>
    <add_on_products config:type="list"/>
  </add-on>
  <dasd>
    <devices config:type="list"/>
    <format_unformatted config:type="boolean">>false</format_unformatted>
  </dasd>
  <deploy_image>
    <image_installation config:type="boolean">>false</image_installation>
  </deploy_image>
  <firewall>
    <enable_firewall config:type="boolean">>false</enable_firewall>
    <start_firewall config:type="boolean">>false</start_firewall>
  </firewall>
  <general>
    <ask-list config:type="list"/>
    <cio_ignore config:type="boolean">>false</cio_ignore>
    <mode>
      <confirm config:type="boolean">>true</confirm>
    </mode>
    <proposals config:type="list"/>
    <signature-handling>
      <accept_file_without_checksum
config:type="boolean">>true</accept_file_without_checksum>

```

```

    <accept_non_trusted_gpg_key
config:type="boolean">true</accept_non_trusted_gpg_key>
    <accept_unknown_gpg_key config:type="boolean">true</accept_unknown_gpg_key>
    <accept_unsigned_file config:type="boolean">true</accept_unsigned_file>
    <accept_verification_failed
config:type="boolean">false</accept_verification_failed>
    <import_gpg_key config:type="boolean">true</import_gpg_key>
</signature-handling>
<storage>
    <partition_alignment
config:type="symbol">align_optimal</partition_alignment>
    <start_multipath config:type="boolean">false</start_multipath>
</storage>
</general>
<kdump>
    <add_crash_kernel config:type="boolean">true</add_crash_kernel>
    <crash_kernel>163M</crash_kernel>
</kdump>
<language>
    <language>en_US</language>
    <languages/>
</language>
<login_settings/>
<networking>
    <dns>
        <dhcp_hostname config:type="boolean">false</dhcp_hostname>
        <write_hostname config:type="boolean">false</write_hostname>
    </dns>
    <ipv6 config:type="boolean">false</ipv6>
    <keep_install_network config:type="boolean">true</keep_install_network>
    <managed config:type="boolean">false</managed>
    <interfaces config:type="list">
        <interface>
            <bootproto>dhcp</bootproto>
            <device>eth0</device>
            <dhclient_set_default_route>yes</dhclient_set_default_route>
            <name>Ethernet Card 0 (virtio4)</name>
            <startmode>auto</startmode>
        </interface>
    </interfaces>
</networking>
<nis>
    <start_autofs config:type="boolean">false</start_autofs>
    <start_nis config:type="boolean">false</start_nis>
</nis>
<ntp-client>
    <start_at_boot config:type="boolean">false</start_at_boot>
    <start_in_chroot config:type="boolean">false</start_in_chroot>
    <sync_interval config:type="integer">5</sync_interval>
    <synchronize_time config:type="boolean">false</synchronize_time>
</ntp-client>
<partitioning config:type="list">
    <drive>
        <device>/dev/vda</device>
        <disklabel>msdos</disklabel>

```

```

<enable_snapshots config:type="boolean">true</enable_snapshots>
<initialize config:type="boolean">true</initialize>
<partitions config:type="list">
  <partition>
    <create config:type="boolean">true</create>
    <crypt_fs config:type="boolean">>false</crypt_fs>
    <filesystem config:type="symbol">ext2</filesystem>
    <format config:type="boolean">true</format>
    <fstopt>acl,user_xattr</fstopt>
    <loop_fs config:type="boolean">>false</loop_fs>
    <mount>/boot/zipl</mount>
    <mountby config:type="symbol">uuid</mountby>
    <partition_id config:type="integer">131</partition_id>
    <partition_nr config:type="integer">1</partition_nr>
    <partition_type>primary</partition_type>
    <resize config:type="boolean">>false</resize>
    <size>auto</size>
  </partition>
  <partition>
    <create config:type="boolean">true</create>
    <crypt_fs config:type="boolean">>false</crypt_fs>
    <filesystem config:type="symbol">swap</filesystem>
    <format config:type="boolean">true</format>
    <fstopt>defaults</fstopt>
    <loop_fs config:type="boolean">>false</loop_fs>
    <mount>swap</mount>
    <mountby config:type="symbol">uuid</mountby>
    <partition_id config:type="integer">130</partition_id>
    <partition_nr config:type="integer">2</partition_nr>
    <resize config:type="boolean">>false</resize>
    <!-- Minimum of 4G of swap required for Oracle 19c installation -->
    <size>4G</size>
  </partition>
  <partition>
    <create config:type="boolean">true</create>
    <crypt_fs config:type="boolean">>false</crypt_fs>
    <filesystem config:type="symbol">btrfs</filesystem>
    <format config:type="boolean">true</format>
    <fstopt>defaults</fstopt>
    <loop_fs config:type="boolean">>false</loop_fs>
    <mount>/</mount>
    <mountby config:type="symbol">uuid</mountby>
    <partition_id config:type="integer">131</partition_id>
    <partition_nr config:type="integer">3</partition_nr>
    <partition_type>primary</partition_type>
    <resize config:type="boolean">>false</resize>
    <size>max</size>
  </partition>
</partitions>
<pesize/>
<type config:type="symbol">CT_DISK</type>
<use>all</use>
</drive>
<drive>
  <device>/dev/vdb</device>

```

```

<disklabel>msdos</disklabel>
<enable_snapshots config:type="boolean">>false</enable_snapshots>
<initialize config:type="boolean">>true</initialize>
<partitions config:type="list">
  <partition>
    <create config:type="boolean">>true</create>
    <crypt_fs config:type="boolean">>false</crypt_fs>
    <filesystem config:type="symbol">xf</filesystem>
    <format config:type="boolean">>true</format>
    <fstopt>defaults</fstopt>
    <loop_fs config:type="boolean">>false</loop_fs>
    <mount>/opt/oracle</mount>
    <mountby config:type="symbol">uuid</mountby>
    <partition_id config:type="integer">131</partition_id>
    <partition_nr config:type="integer">1</partition_nr>
    <partition_type>primary</partition_type>
    <resize config:type="boolean">>false</resize>
    <size>max</size>
  </partition>
</partitions>
<pesize/>
<type config:type="symbol">CT_DISK</type>
<use>all</use>
</drive>
<drive>
  <device>/dev/vdc</device>
  <disklabel>msdos</disklabel>
  <enable_snapshots config:type="boolean">>false</enable_snapshots>
  <initialize config:type="boolean">>true</initialize>
  <partitions config:type="list">
    <partition>
      <create config:type="boolean">>true</create>
      <crypt_fs config:type="boolean">>false</crypt_fs>
      <filesystem config:type="symbol">xf</filesystem>
      <format config:type="boolean">>true</format>
      <fstopt>defaults</fstopt>
      <loop_fs config:type="boolean">>false</loop_fs>
      <mount>/opt/oracle/oradata</mount>
      <mountby config:type="symbol">uuid</mountby>
      <partition_id config:type="integer">131</partition_id>
      <partition_nr config:type="integer">1</partition_nr>
      <partition_type>primary</partition_type>
      <resize config:type="boolean">>false</resize>
      <size>max</size>
    </partition>
  </partitions>
  <pesize/>
  <type config:type="symbol">CT_DISK</type>
  <use>all</use>
</drive>
</partitioning>
<report>
  <errors>
    <log config:type="boolean">>true</log>
    <show config:type="boolean">>true</show>
  </errors>
</report>

```

```

    <timeout config:type="integer">0</timeout>
</errors>
<messages>
    <log config:type="boolean">true</log>
    <show config:type="boolean">true</show>
    <timeout config:type="integer">0</timeout>
</messages>
<warnings>
    <log config:type="boolean">true</log>
    <show config:type="boolean">true</show>
    <timeout config:type="integer">0</timeout>
</warnings>
<yesno_messages>
    <log config:type="boolean">true</log>
    <show config:type="boolean">true</show>
    <timeout config:type="integer">0</timeout>
</yesno_messages>
</report>
<services-manager>
    <default_target>multi-user</default_target>
</services-manager>
<software>
    <image/>
    <install_recommended config:type="boolean">true</install_recommended>
    <instsource/>
    <packages config:type="list">
        <package>vlan</package>
        <package>snapper</package>
        <package>sles-release</package>
        <package>openssh</package>
        <package>lvm2</package>
        <package>kexec-tools</package>
        <package>kdump</package>
        <package>grub2</package>
        <package>glibc</package>
        <package>e2fsprogs</package>
        <package>btrfsprogs</package>
        <!-- Required by ora-val-rpm-S12-DB-19c-19.0.1-1.s390x.rpm for Oracle 19c
install -->
        <package>libaio1-32bit</package>
        <package>libXp6</package>
        <package>libXp6-32bit</package>
    </packages>
    <patterns config:type="list">
        <pattern>32bit</pattern>
        <pattern>Basis-Devel</pattern>
        <pattern>Minimal</pattern>
        <pattern>base</pattern>
        <pattern>documentation</pattern>
        <pattern>oracle_server</pattern>
        <pattern>sles-Basis-Devel-32bit</pattern>
        <pattern>sles-Minimal-32bit</pattern>
        <pattern>sles-base-32bit</pattern>
        <pattern>sles-documentation-32bit</pattern>
        <pattern>sles-oracle_server-32bit</pattern>

```

```
    <pattern>sles-x11-32bit</pattern>
    <pattern>x11</pattern>
    <pattern>yast2</pattern>
  </patterns>
</software>
<timezone>
  <hwclock>UTC</hwclock>
  <timezone>America/New_York</timezone>
</timezone>
<users config:type="list">
  <user>
    <fullname>root</fullname>
    <gid>0</gid>
    <home>/root</home>
    <password_settings>
      <expire/>
      <flag/>
      <inact/>
      <max/>
      <min/>
      <warn/>
    </password_settings>
    <shell>/bin/bash</shell>
    <uid>0</uid>
    <user_password>password</user_password>
    <username>root</username>
  </user>
</users>
<zfcf>
  <devices config:type="list"/>
</zfcf>
</profile>
```

Redbooks

Virtualization Cookbook for IBM Z Volume 5: KVM

SG24-8463-01

ISBN 0738460842



(0.5" spine)

0.475" x 0.873"

250 <-> 459 pages



SG24-8463-01

ISBN 0738460842

Printed in U.S.A.

Get connected

