AlmaLinux 9 Essentials

Payload
publishing

AlmaLinux 9 Essentials

AlmaLinux 9 Essentials

Rev: 1.0

Contents

Table of Contents

Chapter 1

1. Introduction

AlmaLinux 9 Essentials is intended to provide detailed information on the installation, use, and administration of the AlmaLinux 9 distribution. For beginners, the book covers topics such as operating system installation, the basics of the GNOME desktop environment, configuring email and web servers, and installing packages and system updates. Additional installation topics, such as dual booting with Microsoft Windows, are also covered, together with all important security topics, such as configuring a firewall and user and group administration.

For the experienced user, topics such as remote desktop access, the Cockpit web interface, logical volume management (LVM), disk partitioning, swap management, KVM virtualization, Secure Shell (SSH), Linux Containers, and file sharing using both Samba and NFS are covered in detail to provide a thorough overview of this enterprise class operating system.

1.1 Superuser Conventions

AlmaLinux 9, in common with Linux in general, has two types of user accounts, one being a standard user account with restricted access to many of the administrative files and features of the operating system and the other a superuser (*root*) account with elevated privileges. Typically, a user can gain root access either by logging in as the root user or using the *su* - command and entering the root password. In the following example, a user is gaining root access via the *su* - command:

```
[root@demoserver ~]$ su -
Password:
[root@demoserver ~]#
```

Note that the command prompt for a regular user ends with a $ sign while the root user has a # character. When working with the command line, this is a useful indication of whether you are currently issuing commands as the root user.

If the *su* - command fails, the root account on the system has most likely been disabled for security reasons. In this case, the *sudo* command can be used instead, as outlined below.

Using *sudo*, a single command requiring root privileges may be executed by a non-root user. Consider the following attempt to update the operating system with the latest patches and packages:

```
$ dnf update
Error: This command has to be run with superuser privileges (under the root user
on most systems).
```

Optionally, user accounts may be configured so that they have access to root-level privileges. Instead of using the *su* - command to first gain root access, user accounts with administrative privileges are able to run otherwise restricted commands using *sudo*:

```
$ sudo dnf update

We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:

    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.

[sudo] password for demo:
.
.
```

To perform multiple commands without repeatedly using the sudo command, a command prompt with persistent super-user privileges may be accessed as follows:

```
[root@demoserver ~]$ sudo su -
[root@demoserver ~]#
```

The reason for raising this issue so early in the book is that many of the command-line examples outlined in this book will require root privileges. Rather than repetitively preface every command-line example with directions to run the command as root, the command prompt at the start of the line will be used to indicate whether or not the command needs to be performed as root. If the command can be run as a regular user, the command will be prefixed with a $ command prompt as follows:

```
$ date
```

If, on the other hand, the command requires root privileges, the command will be preceded by a # command prompt:

```
# dnf install openssh
```

1.2 Opening a Terminal Window

If you are using the GNOME desktop and need to access a command prompt, you will need to open a Terminal window. To do this, either press the keyboard Windows key or click on the Activities button in the top left-hand corner of the screen, then select the Terminal from the dash as shown in Figure 1-1:

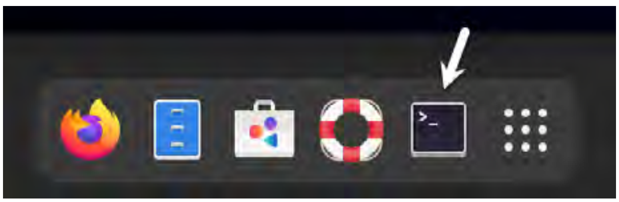

Figure 1-1

1.3 Editing Files

Configuring a Linux system typically involves editing files. For those new to Linux, it can be unclear which editor to use. If you are running a terminal session and do not already have a preferred editor, we recommend using the *nano* editor. To launch *nano* in a terminal window, enter the following command:

```
# nano <file>
```

Where <file> is replaced by the path to the file you wish to edit. For example:

```
# nano /etc/passwd
```

Once loaded, *nano* will appear as illustrated in Figure 1-2:

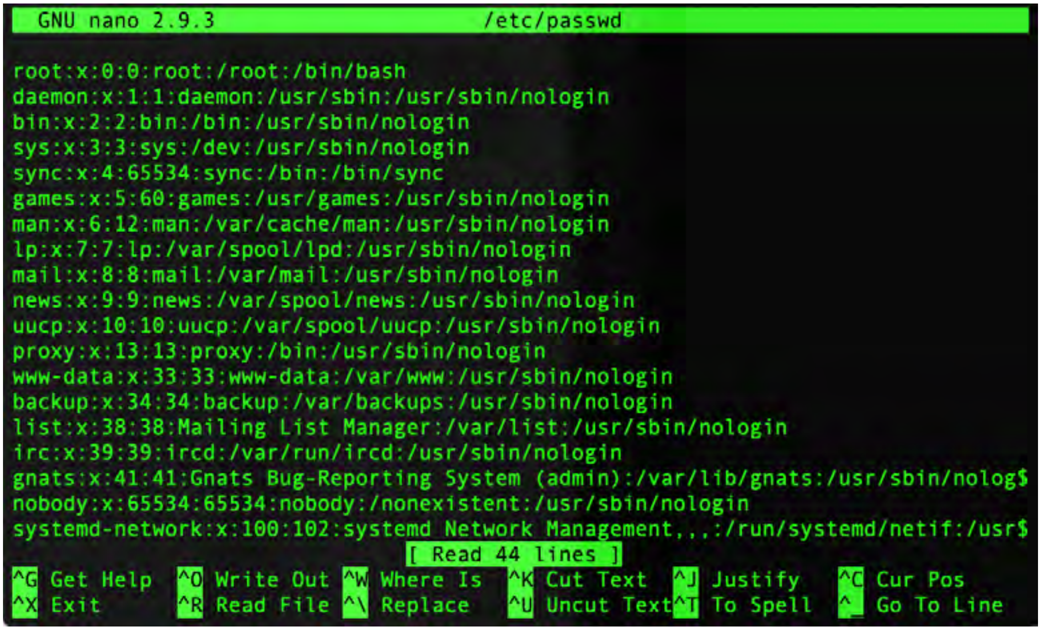

Figure 1-2

To create a new file run *nano* as follows:

```
# nano
```

When you have finished editing the file, type Ctrl-S to save the file, followed by Ctrl-X to exit. To open an existing file, use the Ctrl-R keyboard shortcut.

If you prefer to use a graphical editor within the GNOME desktop environment, *gedit* is a useful starting point for basic editing tasks. To launch *gedit* from the desktop press Alt-F2 to display the Enter a Command window as shown in Figure 1-3:

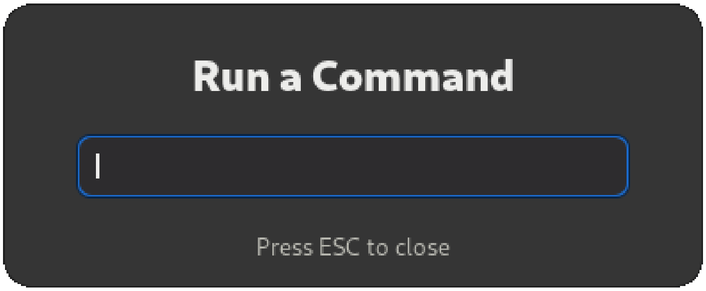

Figure 1-3

Enter *gedit* into the text field and press the Enter key. After a short delay, gedit will load ready to open, create, and edit files:

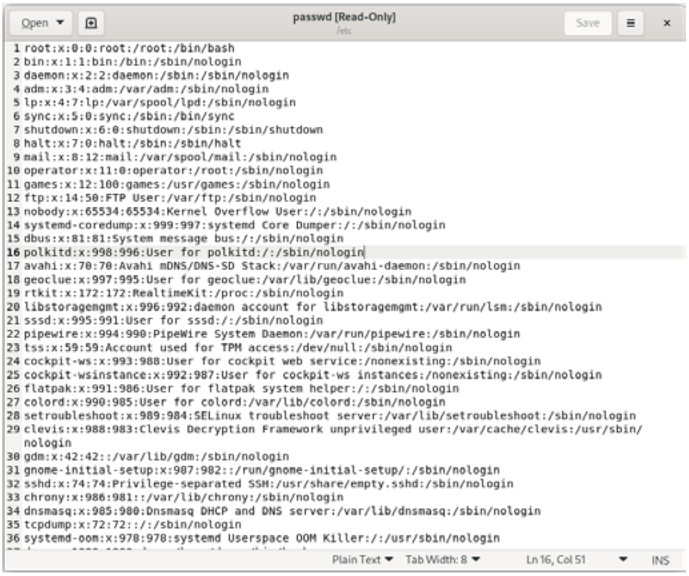

Figure 1-4

Alternatively, launch *gedit* from a terminal window either with or without the path to the file to open:

```
# gedit
# gedit /etc/passwd
```

1.4 Feedback

We want you to be satisfied with your purchase of this book. If you find any errors in the book or have any comments, questions, or concerns, please contact us at *feedback@ebookfrenzy.com*.

1.5 Errata

While we make every effort to ensure the accuracy of the content of this book, it is inevitable that a book covering a subject area of this size and complexity may include some errors and oversights. Any known issues with the book will be outlined, together with solutions, at the following URL:

https://www.ebookfrenzy.com/errata/almalinux9.html

In the event that you find an error not listed in the errata, please let us know by emailing our support team at *feedback@ebookfrenzy.com*.

Chapter 2

2. A Brief History of AlmaLinux

AlmaLinux 9 is one of several variants (also referred to as *distributions*) of the Linux operating system. It is based on the source code of the Red Hat Enterprise Linux distribution (RHEL), developed by a U.S. company named Red Hat, Inc. Based in Raleigh, North Carolina, the company was founded in the mid-1990s through the merger of two companies owned at the time by Marc Ewing and Bob Young. The origins of Linux, however, go back even further. This chapter will outline the history of both the Linux operating system and Red Hat, Inc. before explaining how AlmaLinux fits into this picture.

2.1 What exactly is Linux?

Linux is an operating system in much the same way that Windows is an operating system (and there any similarities between Linux and Windows end). The term operating system is used to describe the software that acts as a layer between the hardware in a computer and the applications that we all run on a daily basis. When programmers write applications, they interface with the operating system to perform such tasks as writing files to the hard disk drive and displaying information on the screen. Without an operating system, every programmer would have to write code to access the hardware of the system directly. In addition, the programmer would have to be able to support every single piece of hardware ever created to be sure the application would work on every possible hardware configuration. Because the operating system handles all of this hardware complexity, application development becomes a much easier task. Linux is just one of a number of different operating systems available today.

2.2 UNIX Origins

To understand the history of Linux, we first have to go back to AT&T Bell Laboratories in the late 1960s. During this time, AT&T had discontinued involvement in developing a new operating system named Multics. However, two AT&T engineers, Ken Thompson, and Dennis Ritchie, decided to take what they had learned from the Multics project and create a new operating system named UNIX which quickly gained popularity and wide adoption both with corporations and academic institutions.

A variety of proprietary UNIX implementations eventually came to market, including those created by IBM (AIX), Hewlett-Packard (HP-UX), and Sun Microsystems (SunOS and Solaris). In addition, a UNIX-like operating system named MINIX was created by Andrew S. Tanenbaum and designed for educational use with source code access provided to universities.

2.3 Who Created Linux?

The origins of Linux can be traced back to the work and philosophies of two people. At the heart of the Linux operating system is something called the *kernel*. This is the core set of features necessary for the operating system to function. The kernel manages the system's resources and handles communication between the hardware and the applications. The Linux kernel was developed by Linus Torvalds, who, taking a dislike to MS-DOS and impatient for the availability of MINIX for the new Intel 80386 microprocessor, decided to write his own UNIX-like kernel. When he had finished the first version of the kernel, he released it under an open-source license that enabled anyone to download the source code and freely use and modify it without having to pay Linus any money.

Around the same time, Richard Stallman at the Free Software Foundation, a strong advocate of free and open-source software, was working on an open-source operating system of his own. Rather than focusing initially on the kernel, however, Stallman began by developing open-source versions of all the UNIX tools, utilities, and compilers necessary to use and maintain an operating system. By the time he had finished developing this infrastructure, the obvious solution was to combine his work with the kernel Linus had written to create a complete operating system. This combination became known as GNU/Linux. Purists insist that Linux always be referred to as GNU/Linux (in fact, at one time, Richard Stallman refused to give press interviews to any publication which failed to refer to Linux as GNU/Linux). This is not unreasonable, given that the GNU tools developed by the Free Software Foundation make up a significant and vital part of GNU/Linux. Unfortunately, most people and publications refer to Linux as Linux, which will probably always continue to be the case.

2.4 The Early Days of Red Hat

In 1993 Bob Young created a company named ACC Corporation which, according to Young, he ran from his "wife's sewing closet". The name ACC was intended to represent a catalog business but was also an abbreviation of a small business his wife ran called "Antiques and Collectibles of Connecticut". Among the items sold through the ACC catalog business were Linux CDs and related open-source software.

Around the same time, Marc Ewing had created his own Linux distribution company, which he named Red Hat Linux (after his propensity to wear a red baseball cap while at Carnegie Mellon University).

In 1995, ACC acquired Red Hat, adopted the name Red Hat, Inc., and experienced rapid and significant growth. Bob Young stepped down as CEO shortly after the company went public in August of 1999 and has since pursued a number of business and philanthropic efforts, including a print-on-demand book publishing company named Lulu and ownership of two Canadian professional sports teams. In 2018, IBM acquired Red Hat, Inc. in a deal valued at $34 billion.

2.5 Red Hat Support

Early releases of Red Hat Linux were shipped to customers on floppy disks and CDs (this, of course, predated the widespread availability of broadband internet connections). When users encountered problems with the software, they could only contact Red Hat by email. In fact, Bob Young often jokes that this was effective in limiting support requests since, by the time a customer realized they needed help, their computer was usually inoperative and therefore unavailable to be used to send an email message seeking assistance from Red Hat's support team. In later years, Red Hat provided better levels of support tied to paid subscriptions and now provides a variety of support levels ranging from "self-help" (no support) up to premium support.

2.6 Open Source

Red Hat Enterprise Linux 9 is the current commercial offering from Red Hat and is primarily targeted at corporate, mission-critical installations. It is also the cornerstone of an expanding ecosystem of products and services Red Hat offers.

RHEL used to be an open-source product in that anyone could download the source code free of charge and build the software themselves (a task not to be undertaken lightly). That changed in 2023 when Red Hat began making the source code available to paying customers only.

2.7 The Fedora Project

Red Hat also sponsors the Fedora Project, the goal of which is to provide access to a free Linux operating system (in both source and binary distributions) in the form of Fedora Linux. Fedora Linux also serves as a proving ground for many of the new features that are eventually adopted into the Red Hat Enterprise Linux operating system family and the CentOS derivative.

2.8 CentOS Stream

For users unable to afford a Red Hat Enterprise Linux subscription, another option is the CentOS Stream operating system. The CentOS project, initially a community-driven effort but now owned by Red Hat, takes the Red Hat Enterprise Linux source code, removes the Red Hat branding and subscription requirements, compiles it, and provides the distribution for download. Like Fedora, CentOS Stream field tests new operating system features before they are included in a future RHEL release. As such, it may lack stability but provides access to cutting-edge features.

2.9 AlmaLinux

AlmaLinux was created in 2021 by the AlmaLinux OS Foundation to provide a stable Linux distribution that is 100% compatible with Red Hat Enterprise Linux. This originally involved building from the RHEL source code. Now that the RHEL source code is no longer publicly available, the goal is changing to 100% binary compatibility. Binary compatibility means that while AlmaLinux may not be identical to RHEL, it can run the same software and applications that run on RHEL.

2.10 Summary

The origins of the Linux operating system can be traced back to the work of Linus Torvalds and Richard Stallman in the form of the Linux kernel combined with the tools and compilers built by the GNU project.

Over the years, the open-source nature of Linux has resulted in the release of a wide range of different Linux distributions. One such distribution is Red Hat Enterprise Linux, created by Red Hat, Inc., founded by Bob Young and Mark Ewing. Red Hat specializes in providing enterprise-level Linux software solutions combined with extensive technical support services. AlmaLinux is based on and 100% binary compatible with Red Hat Enterprise Linux.

Chapter 4

4. Dual Booting AlmaLinux 9 with Windows

Like most Linux distributions, AlmaLinux 9 will happily co-exist on a hard disk drive with just about any version of Windows up to and including Windows 11. This is a concept known as dual-booting. When you power up the system, you will be presented with a menu providing the option to boot either your AlmaLinux 9 installation or Windows. Obviously, you can only run one operating system at a time. Still, it is worth noting that the files on the Windows partition of your disk drive will be available to you from AlmaLinux 9 regardless of whether your Windows partition was formatted using NTFS, FAT16, or FAT32.

This installation method involves shrinking the size of the existing Windows partitions and then installing AlmaLinux 9 into the reclaimed space. This chapter will assume that AlmaLinux 9 is being installed on a system currently running Windows 11.

4.1 Partition Resizing

To accommodate AlmaLinux 9 on a disk drive that already contains a Windows installation, the first step involves shrinking the Windows partition to make some room. The recommended course of action is to use the Windows Disk Management interface to reduce the partition size before attempting to install AlmaLinux 9.

To access Disk Management on Windows 11, right-click on the Start menu and select the option from the resulting menu as highlighted in Figure 4-1:

Figure 4-1

Once loaded, the Disk Management tool will display a graphical representation of the disk drives detected on the system:

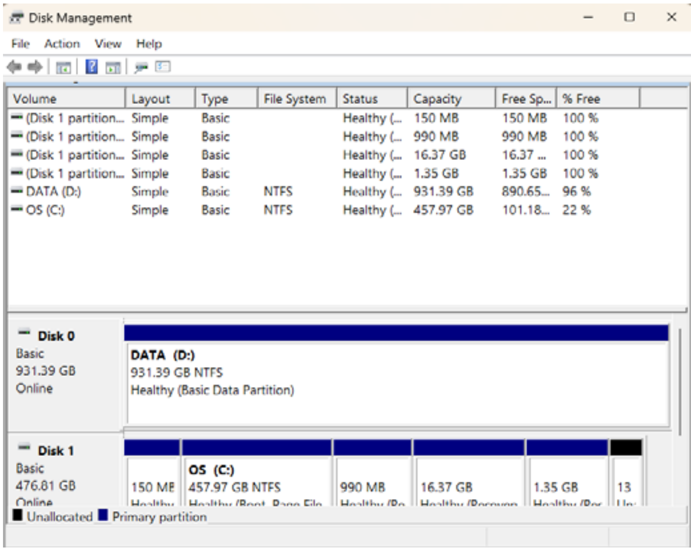

Figure 4-2

Right-click on the partition you wish to reduce in size and select *Shrink Volume...* from the popup menu. The tool will calculate the maximum amount by which the volume size can be reduced without data loss (a process that can take several minutes depending on the overall size of the partition). Once this analysis is complete, a dialog similar to the one in Figure 4-3 below will appear:

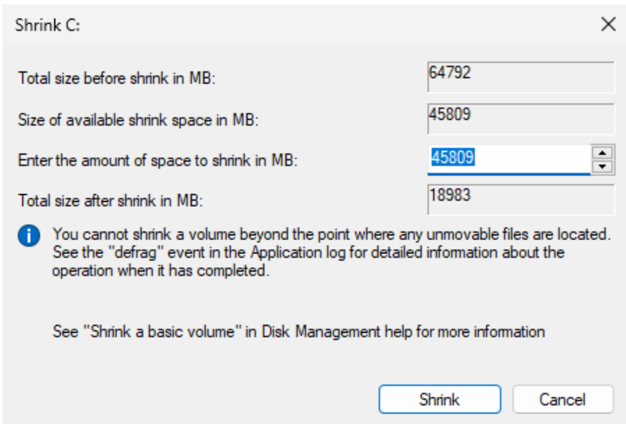

Figure 4-3

Specify a value in the *Enter amount of space to shrink in the MB* field and click the *Shrink* button to proceed. Once the resizing operation is complete, reboot using the AlmaLinux 9 installation media (as outlined in *"Installing AlmaLinux 9 on a Clean Disk Drive"*) and install using the new free space. During the AlmaLinux 9 installation process, this can be achieved by selecting the Installation Destination option on the Installation Summary screen and ensuring that the Automatic storage configuration option is selected. This will automatically install AlmaLinux 9 into the unallocated space created when the Windows partition was reduced in size.

Once installation of AlmaLinux 9 onto the disk is complete and the system has restarted, the standard AlmaLinux 9 boot menu will appear, including an additional option to boot the Windows system:

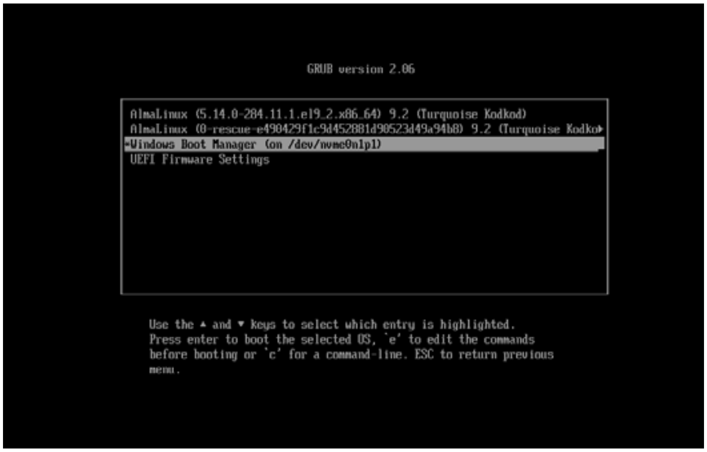

Figure 4-4

4.2 Changing the Default Boot Option

When the system starts, the boot options screen will appear, and wait 5 seconds for the user to choose an operating system. If no selection has been made before the timeout elapses, the default operating system will be started. The default operating system will be the standard (non-rescue) AlmaLinux 9 image on a newly configured system. This default can, however, be changed from within AlmaLinux 9.

A range of boot configuration options (including the 5-second timeout and the boot RHGB settings outlined in *"Installing AlmaLinux 9 on a Clean Disk Drive"*) are declared in the */etc/default/grub* file, which reads as follows on a new installation:

```
GRUB_TIMEOUT=5
GRUB_DISTRIBUTOR="$(sed 's, release .*$,,g' /etc/system-release)"
GRUB_DEFAULT=saved
GRUB_DISABLE_SUBMENU=true
GRUB_TERMINAL_OUTPUT="console"
GRUB_CMDLINE_LINUX="crashkernel=1G-4G:192M,4G-64G:256M,64G-:512M resume=/dev/
mapper/almalinux-swap rd.lvm.lv=almalinux/root rd.lvm.lv=almalinux/swap rhgb
quiet"
GRUB_DISABLE_RECOVERY="true"
GRUB_ENABLE_BLSCFG=true
```

The first step in changing the default boot system is to declare the GRUB_SAVEDEFAULT setting within this file:

```
GRUB_TIMEOUT=5
GRUB_DISTRIBUTOR="$(sed 's, release .*$,,g' /etc/system-release)"
GRUB_DEFAULT=saved
GRUB_SAVEDEFAULT=true
.
.
```

This setting saves a new default value within the boot configuration. Next, run the *grub2-set-default* command to change the default setting using a numbering system that counts the first option as 0. For example, if the Windows 11 option is position 3 in the menu, the command to make Windows 11 the default boot option would read as follows:

```
# grub2-set-default 2
```

Check that the new setting has taken effect by running the following command:

```
# grub2-editenv list
saved_entry=2
menu_auto_hide=1
boot_success=1
boot_indeterminate=0
```

Note that the *saved_entry* value is now set to the Linux boot partition. After changing the default, regenerate the boot configuration file as follows:

```
# grub2-mkconfig --output=/boot/grub2/grub.cfg
```

Reboot the system and verify that the boot menu defaults to the Windows option and that Windows loads after the timeout expires.

4.3 Accessing the Windows Partition from AlmaLinux 9

When running AlmaLinux 9 in a dual boot configuration, it is possible to access files located on the Windows partition by manually mounting the partition from the command line. Before doing so, however, some additional packages need to be installed on the system. First, the fuse kernel module needs to be downloaded and installed:

```
# dnf install fuse
# modprobe fuse
```

Next, the Fuse NTFS driver needs to be installed. Unfortunately, this package is not included in the standard AlmaLinux 9 repositories, so the Extra Packages for Enterprise Linux (EPEL) repository needs to be added to the system as follows:

```
# dnf config-manager --set-enabled crb
# crb enable
# dnf install \
    https://dl.fedoraproject.org/pub/epel/epel-release-latest-9.noarch.rpm \
    https://dl.fedoraproject.org/pub/epel/epel-next-release-latest-9.noarch.rpm
```

With the EPEL repository added, the driver can now be installed:

```
# dnf install ntfs-3g
```

Once the requisite packages are installed, the next step is to create a directory to use as the mount point for our Windows partition. In this example, we will create a directory named */mnt/windows*:

```
# mkdir /mnt/windows
```

To identify the device name that has been assigned to the Windows partition, use the *fdisk* command as follows:

```
# fdisk -l
.
.
Device              Start       End  Sectors  Size Type
/dev/nvme0n1p1       2048    206847   204800  100M EFI System
/dev/nvme0n1p2     206848    239615    32768   16M Microsoft reserved
/dev/nvme0n1p3     239616  49362943 49123328 23.4G Microsoft basic data
/dev/nvme0n1p4 132933632 134213631  1280000  625M Windows recovery environment
/dev/nvme0n1p5  49362944  51460095  2097152    1G Linux filesystem
/dev/nvme0n1p6  51460096 132933631 81473536 38.8G Linux LVM
```

In the above output, the main Windows partition containing the files we need access to is represented by */dev/nvme0n1p3*. Next, we need to run the *mount* command (assuming the Windows partition is */dev/nvme0n1p3*) as follows:

```
# mount /dev/nvme0n1p3 /mnt/windows
```

Check that the mount was successful by listing the contents of the top-level directory of the mount point:

```
# ls /mnt/windows
'$Recycle.Bin'              ProgramData             swapfile.sys
'Documents and Settings'   'Program Files'         'System Volume Information'
 pagefile.sys              'Program Files (x86)'    Users
 PerfLogs                   Recovery                Windows
```

To automate the mount each time the system is booted, add the appropriate mount line to the */etc/fstab* file:

```
/dev/nvme0n1p3 /mnt/windows ntfs defaults 0 0
```

To unmount the Windows file system at any time:

```
# umount /mnt/windows
```

4.4 Summary

AlmaLinux 9 can safely co-exist on the same disk drive as a Windows operating system by creating a dual boot environment. This involves shrinking the Windows system's space to make room for AlmaLinux 9 before performing the installation. Once AlmaLinux 9 has been installed, the boot menu configuration must be modified to include the option to boot from Windows. To access the Windows filesystem from within AlmaLinux 9, the Fuse NTFS driver must be installed and used to mount the Windows partitions.

Chapter 7

7. An Overview of the Cockpit Web Interface

Although equipped with the latest Linux desktop environment, AlmaLinux 9 is very much a server operating system. As such, most AlmaLinux deployments will be to remote physical servers or as cloud-based virtual machine instances. Invariably, these systems run without a keyboard, mouse, or monitor, with direct access only available via the command prompt over a network connection. This presents a challenge in terms of administering the system from remote locations. While much can certainly be achieved via remote access to the command-line and desktop environments, there needs to be a consistent and cohesive solution to the administrative and monitoring tasks that must be performed daily on an enterprise-level operating system such as AlmaLinux 9.

The Cockpit web-based administration interface provides this functionality. This chapter will explain how to install, configure and access the Cockpit interface while also providing an overview of the key features of Cockpit, many of which will be covered in greater detail in later chapters.

7.1 An Overview of Cockpit

Cockpit is a lightweight, web-based interface that allows general system administrative tasks to be performed remotely. When installed and configured, the system administrator opens a local browser window and navigates to the Cockpit port on the remote server. After loading the Cockpit interface into the browser and logging in, a wide range of tasks can be performed visually using administration and monitoring tools.

Behind the scenes, Cockpit uses the same tools to perform tasks typically used when working at the command line and updates automatically to reflect changes occurring elsewhere on the system. This allows Cockpit to be used with other administration tools and techniques without the risk of one approach overriding another. Cockpit can also be configured to access more than one server, allowing multiple servers to be administered and monitored simultaneously through a single browser session.

Cockpit is installed by default with a wide range of tools already bundled and allows additional extension plugins to be installed as needed. Cockpit is also designed so that you can create your own extensions using a combination of HTML and JavaScript to add missing or custom functionality.

Cockpit's modular design also allows many features to be embedded into other web-based applications.

7.2 Installing and Enabling Cockpit

Cockpit is generally not installed on AlmaLinux 9 by default but can be set up and enabled in a few simple steps. The first step is to install the Cockpit package as follows:

```
# dnf install cockpit
```

Next, the Cockpit socket service needs to be enabled:

```
# systemctl enable --now cockpit.socket
```

Finally, the necessary ports need to be opened on the firewall to allow remote browser connections to reach Cockpit if a firewall is enabled on your system (for details on firewalls, refer to the chapter entitled *"AlmaLinux 9 Firewall Basics"*).

```
# firewall-cmd --add-service=cockpit --permanent
# firewall-cmd --reload
```

7.3 Accessing Cockpit

If you have access to the desktop environment of the server on which Cockpit has been installed, open a browser window and navigate to *https://localhost:9090* to access the Cockpit sign-in screen. If, on the other hand, the server is remote, navigate to the server using the domain name or IP address (for example, *https://myserver.com*:9090).

When the connection is established, the browser may warn that the connection is not secure. This is because the Cockpit service uses a self-signed certificate. Select the option to proceed to the website or, to avoid this message in the future, select the advanced option and add an exception for the server address. Once connected, the browser will load the login page shown in Figure 7-1 below:

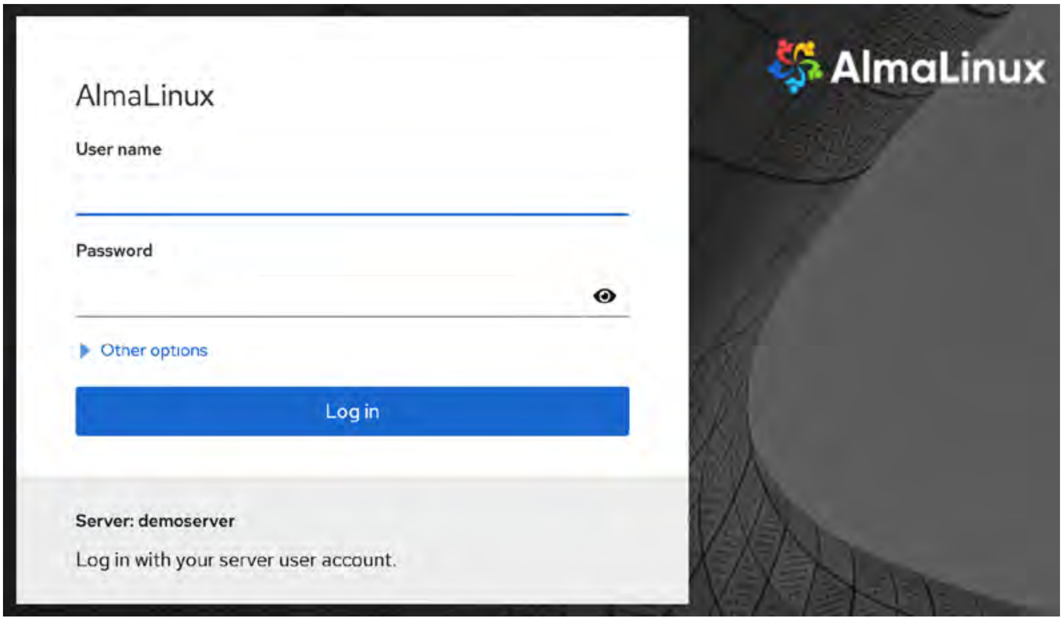

Figure 7-1

Sign in to the Cockpit interface either as root or with your user account credentials. Note that some tasks will be restricted within the Cockpit interface when signed in as a user due to permission constraints. In this situation, the Cockpit console will display a button labeled "Limited Access," as shown in Figure 7-2:

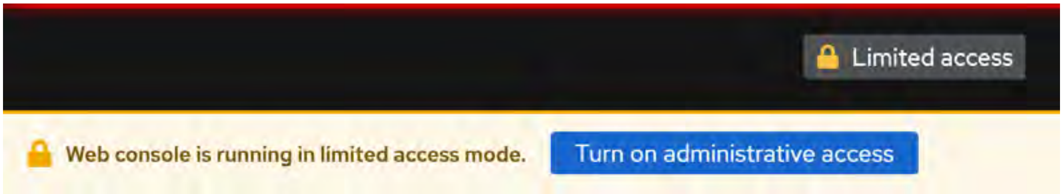

Figure 7-2

To elevate your privileges, click on the limited access button and enter your password when you are prompted to do so:

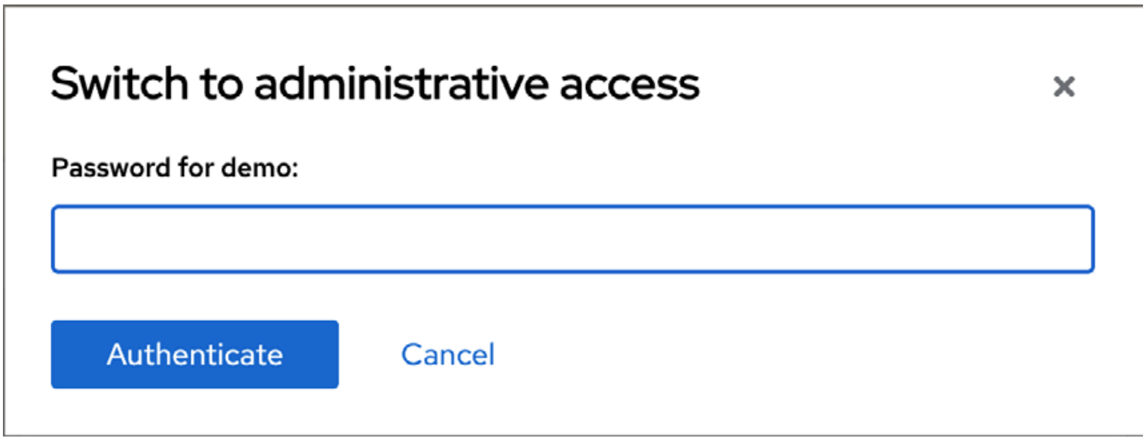

Figure 7-3

After signing in, Cockpit will display the Overview screen.

7.4 Overview

The Overview screen provides an overview of the current system, including access to CPU, memory, storage, and network activity performance metrics. This screen also includes information about the system, including the underlying hardware, hostname, system time, and whether the system software is up to date. Options are also provided to restart or shut down the system.

Figure 7-4, for example, shows the Overview page of the Cockpit interface:

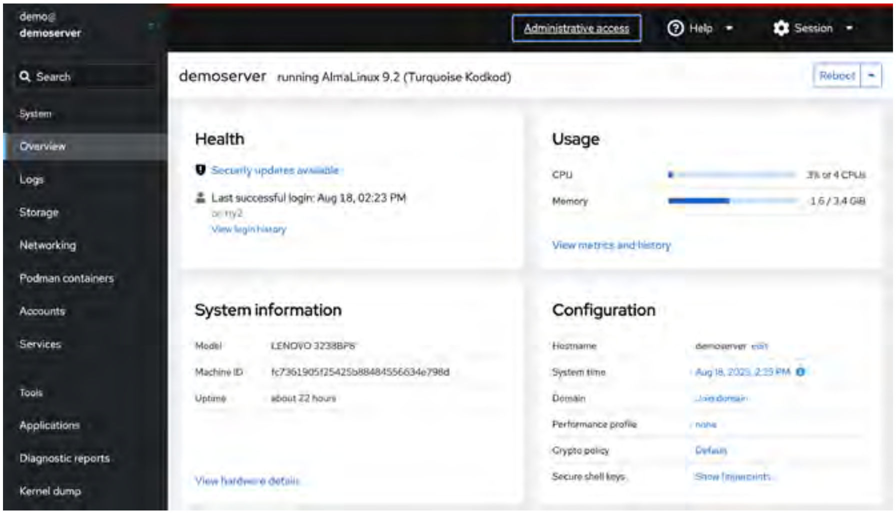

Figure 7-4

For more information on a particular category, click on the corresponding link. Figure 7-5, for example, shows the system performance history screen:

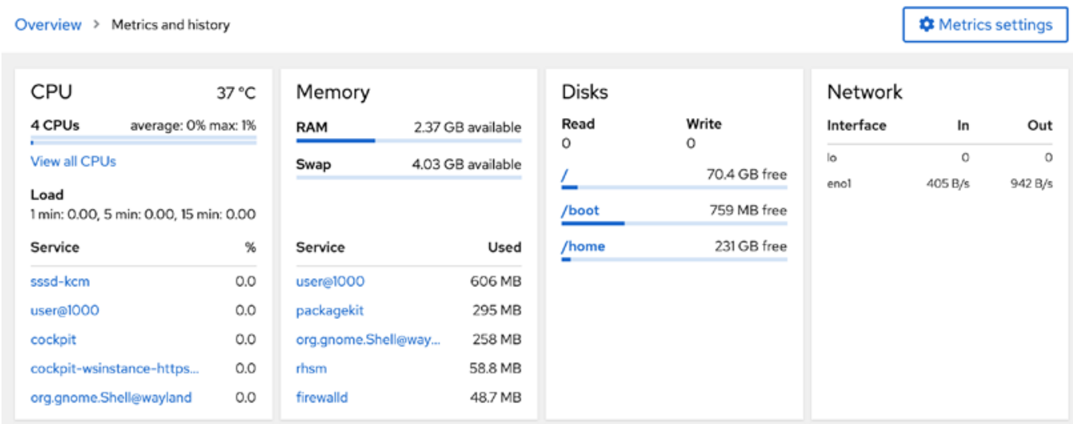

Figure 7-5

7.5 Logs

When the Logs category is selected, Cockpit displays the contents of the *systemd* journal logs. Choosing a log entry will display the entire log message. The log entries are ordered with the most recent at the top, and menus are included to filter the logs for different time durations and based on message severity.

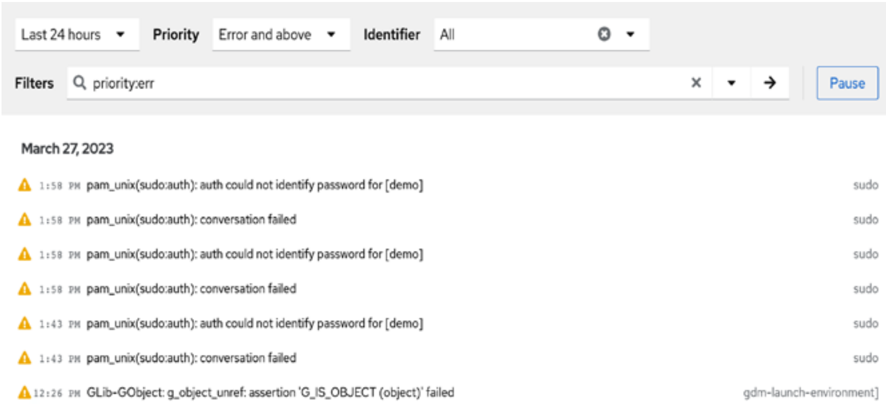

Figure 7-6

7.6 Storage

Select the Storage option to review and manage the storage on the system, including disks, partitions, volume groups, Network File System (NFS) mounts, and RAID storage. This screen also allows disk I/O activity to be monitored in real-time and lists log output from the system *udisksd* service used to query and manage storage devices:

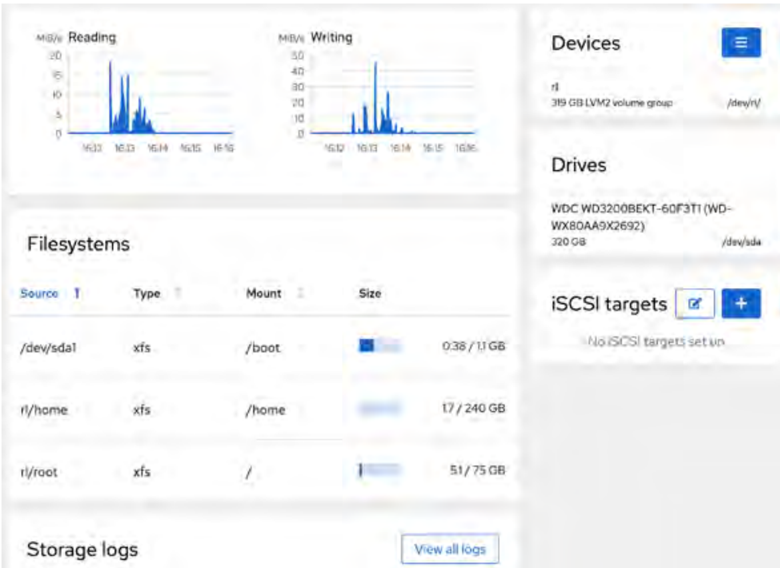

Figure 7-7

7.7 Networking

The Networking screen provides information on various network-related configurations and services, including network interfaces and firewall settings. In addition, it allows configuration changes such as creating network bridges or setting up virtual networks:

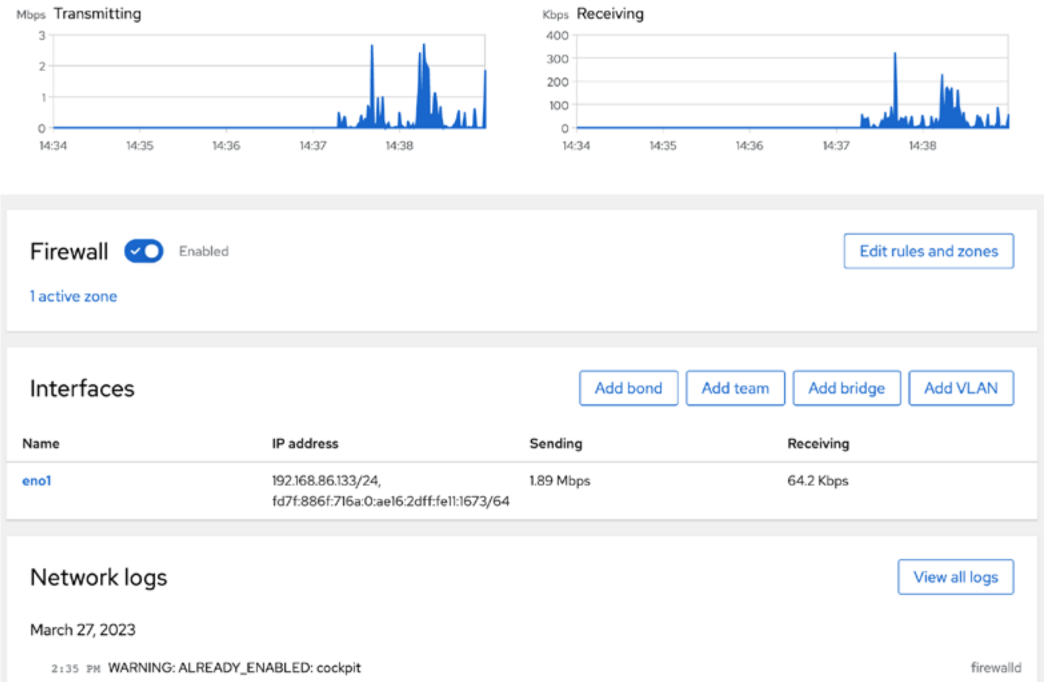

Figure 7-8

7.8 Accounts

Select this option to view the current user accounts configured on the system and create accounts for additional users. The topic of user management will be covered later in the chapter entitled *"Managing AlmaLinux 9 Users and Groups"*:

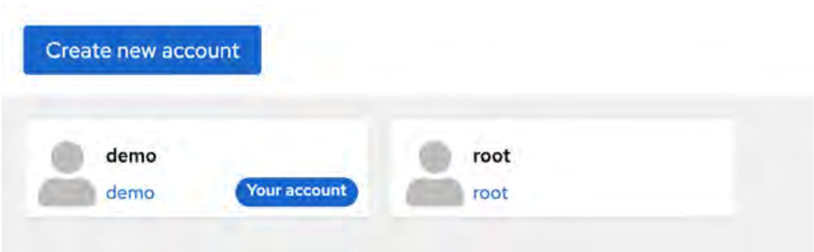

Figure 7-9

Click on an existing account to view details and make changes. The user account details page may also be used to review and add Public SSH keys to the user's account for remote access to the server, as outlined in the chapter *"Configuring SSH Key-based Authentication on AlmaLinux 9"*.

7.9 Services

This screen displays a list of the system services running on the server and allows those services to be added, removed, stopped, and started.

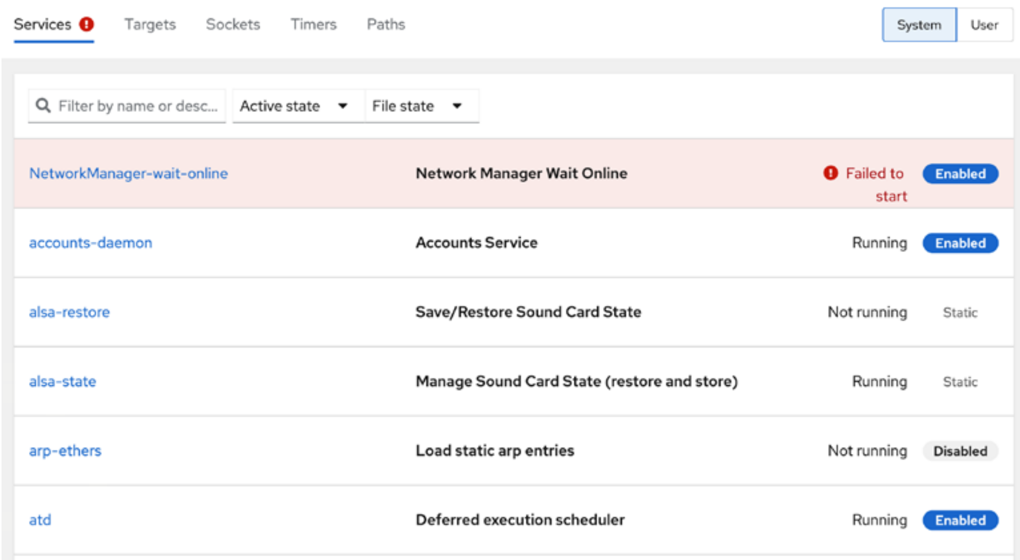

Figure 7-10

The topic of services will be covered in detail in the chapter *"Managing AlmaLinux 9 systemd Units"*.

7.10 Applications

As previously mentioned, additional functionality can be added to Cockpit as extensions. These can either be self-developed extensions or those provided by third parties. The Applications screen lists installed extensions and allows extensions to be added or removed:

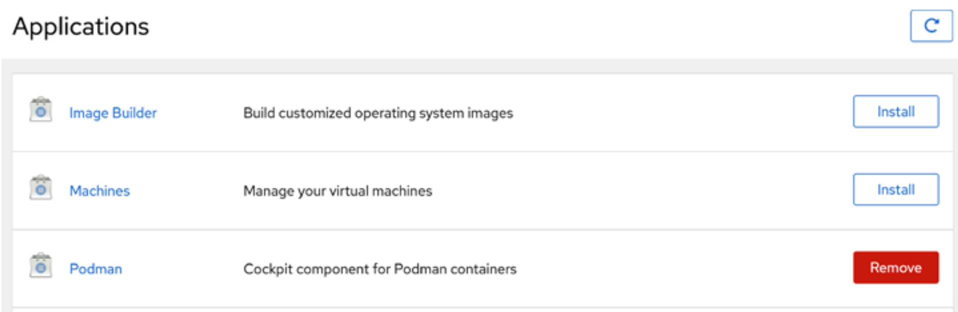

Figure 7-11

7.11 Virtual Machines

Virtualization allows multiple operating system instances to run simultaneously on a single computer system, with each system running inside its own *virtual machine*. The Virtual Machines Cockpit extension provides a way to create and manage the virtual machine guests installed on the server:

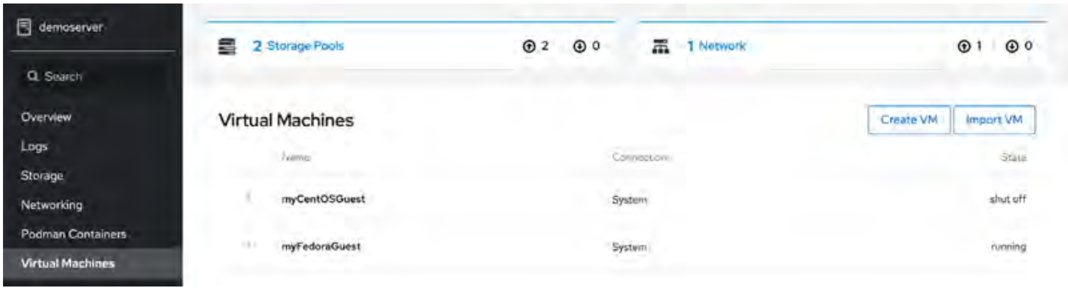

Figure 7-12

The Virtual Machines extension is not installed by default but can be added via the Cockpit Applications screen or by running the following command:

```
# dnf install cockpit-machines
```

The use of virtualization with AlmaLinux 9 is covered starting with the chapter *"An Overview of Virtualization Techniques"*.

7.12 Software Updates

If any software updates are available for the system, they will be listed here and can be installed from this screen:

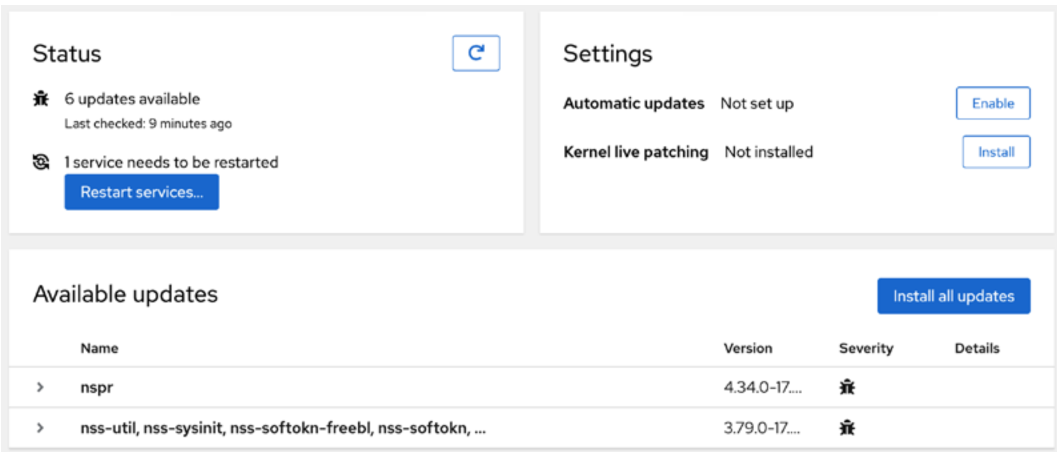

Figure 7-13

7.13 Terminal

As the name suggests, the Terminal screen provides access to the command-line prompt:

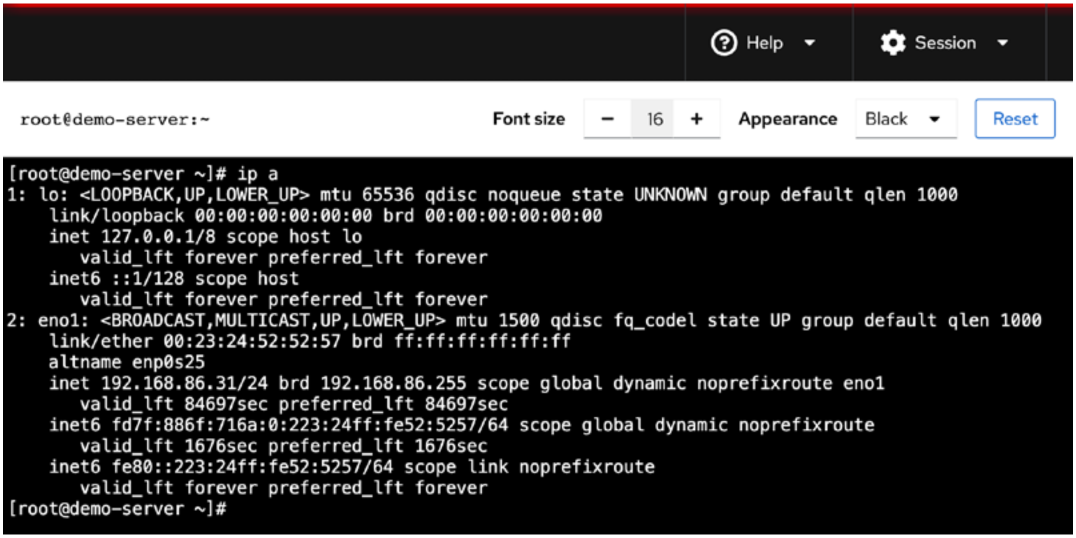

Figure 7-14

7.14 Connecting to Multiple Servers

Cockpit can be configured to administer multiple servers from within a single session. To add another host to the Cockpit session, click on the button highlighted in Figure 7-15 to display the Hosts panel:

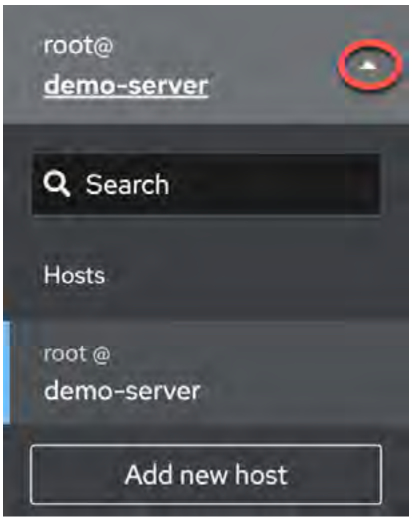

Figure 7-15

Click the *Add new host* button and enter the IP address or hostname of the other system and select a color by which to distinguish this server from any others added to Cockpit before clicking on the Add button:

Figure 7-16

Cockpit will ask you to accept a new SSH key if you are connecting to the remote server for the first time. After accepting the key, you will be prompted to enter the password for the user name specified in Figure 7-16 above. The option is also provided to set up and authorize a password-protected SSH key to enable automatic login to the second host system next time you need to access it:

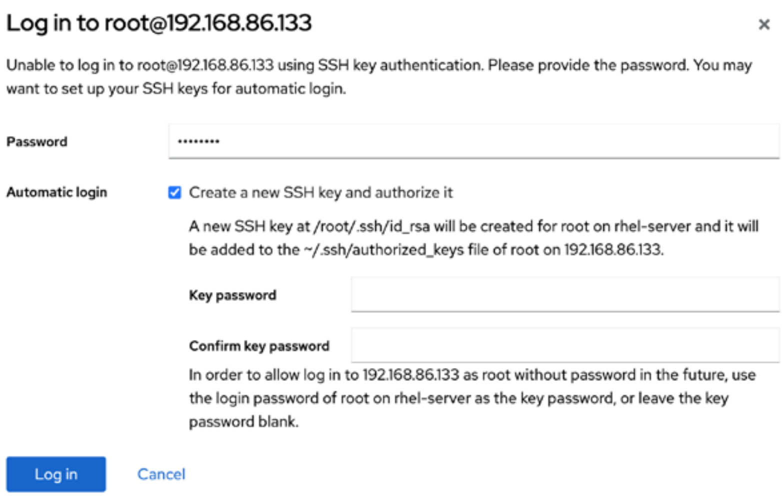

Figure 7-17

To switch between the hosts, display the Hosts panel (Figure 7-15 above) and select the required system.

7.15 Enabling Stored Metrics

In a standard installation, Cockpit does not retain any performance metric data beyond what is displayed in the short time window covered by the graphs. To retain the data collected by Cockpit, the Cockpit Co-Pilot (PCP) package needs to be installed. Begin by installing the *cockpit-pcp* package as follows:

```
# dnf install cockpit-pcp
```

After installing cockpit-pcp, log out of the current Cockpit session and back in.

Next, display the Performance Metrics screen and click on the *Metrics settings* button to display the screen shown in Figure 7-18, switch on the *Collect metrics* option, and click Save:

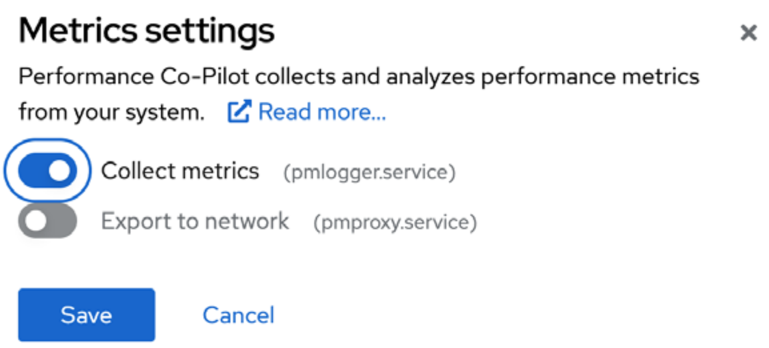

Figure 7-18

After sufficient time has elapsed for Cockpit to gather data, the metric information will appear as shown in Figure 7-19, categorized in hourly blocks:

Figure 7-19

7.16 Summary

The Cockpit web interface allows remote system administration tasks to be performed visually from within a web browser without relying on the command prompt and command-line tools. Once installed and enabled, the system administrator opens a web browser, connects to the remote server, and signs into the Cockpit interface. Behind the scenes, Cockpit uses the same command-line tools as those available via the command prompt, allowing both options to be used without the risk of configuration conflicts. In addition, Cockpit uses a modular framework enabling additional extensions to be added and for custom extensions to be developed and integrated. A Cockpit session can be used to administer a single server or configured to access multiple servers simultaneously.

Chapter 16

16. AlmaLinux 9 Remote Desktop Access with VNC

AlmaLinux 9 can be configured to provide remote access to the graphical desktop environment over a network or internet connection. Although not enabled by default, displaying and accessing an AlmaLinux 9 desktop from a system anywhere else on a network or the internet is relatively straightforward. This can be achieved regardless of whether that system runs Linux, Windows, or macOS. There are even apps available for Android and iOS that will allow you to access your AlmaLinux 9 desktop from just about anywhere that a data signal is available.

Remote desktop access can be helpful in many scenarios. For example, it enables you or another person to view and interact with your AlmaLinux 9 desktop environment from another computer system on the same network or over the internet. This is useful if you need to work on your computer when you are away from your desk, such as while traveling. It is also helpful when a co-worker or IT support technician needs access to your desktop to resolve a problem.

When the AlmaLinux 9 system runs on a cloud-based server, it also allows access to the desktop environment as an alternative to performing administrative tasks using the command-line prompt or Cockpit web console.

The AlmaLinux 9 remote desktop functionality is based on a technology known as Virtual Network Computing (VNC). This chapter will cover the key aspects of configuring and using remote desktops within AlmaLinux 9.

16.1 Secure and Insecure Remote Desktop Access

In this chapter, we will cover both secure and insecure remote desktop access methods. Assuming you are accessing one system from another within a secure internal network, using the insecure access method is generally safe. If, on the other hand, you plan to access your desktop remotely over any public network, you must use the secure method of access to avoid your system and data being compromised.

16.2 Installing the GNOME Desktop Environment

It is, of course, only possible to access the desktop environment if the desktop itself has been installed. If, for example, the system was initially configured as a server, it is unlikely that the desktop packages were installed. The easiest way to install the packages necessary to run the GNOME desktop is to perform a group install. The key to installing groups of packages to enable a specific feature is knowing the group's name. At the time of writing, there are two groups for installing the desktop environment on AlmaLinux 9: "Server with GUI" and "Workstation". As the group names tend to change from one AlmaLinux release to another, it is helpful to know that the list of groups that are either installed or available to be installed can be obtained using the *dnf* utility as follows:

```
# dnf grouplist
Available Environment Groups:
   Server
   Minimal Install
   Workstation
   Virtualization Host
   Custom Operating System
Installed Environment Groups:
   Server with GUI
Installed Groups:
   Container Management
   Headless Management
Available Groups:
   RPM Development Tools
   .NET Development
   Console Internet Tools
   Scientific Support
   Legacy UNIX Compatibility
   Graphical Administration Tools
   Network Servers
   System Tools
   Development Tools
   Security Tools
   Smart Card Support
```

The Workstation environment group is listed as available (and therefore not already installed) in the above example. To find out more information about the contents of a group before installation, use the following command:

```
# dnf groupinfo workstation
Environment Group: Workstation
 Description: Workstation is a user-friendly desktop system for laptops and PCs.
 Mandatory Groups:
   Common NetworkManager submodules
   Core
   Fonts
   GNOME
   Guest Desktop Agents
   Hardware Support
   Internet Browser
   Multimedia
   Printing Client
   Standard
```

```
   Workstation product core
   base-x
 Optional Groups:
   Backup Client
   GNOME Applications
   Headless Management
   Internet Applications
   Office Suite and Productivity
   Remote Desktop Clients
   Smart Card Support
```

Having confirmed that this is the correct group, it can be installed as follows:

```
# dnf groupinstall workstation
```

Once installed, and assuming that the system has a display added, the desktop can be launched using the following *startx* command:

```
$ startx
```

To launch the graphical desktop each time the system starts, change the default target as follows:

```
# systemctl set-default graphical.target
```

If, on the other hand, the system is a server with no directly connected display, the only way to run and access the desktop will be to configure VNC support on the system.

16.3 Installing VNC on AlmaLinux 9

Access to a remote desktop requires a VNC server installed on the remote system, a VNC viewer on the system from which access is being established, and, optionally, a secure SSH connection. While several VNC server and viewer implementations are available, Red Hat has standardized on TigerVNC, which provides both server and viewer components for Linux-based operating systems. VNC viewer clients for non-Linux platforms include RealVNC and TightVNC.

To install the TigerVNC server package on AlmaLinux 9, run the following command:

```
# dnf install tigervnc-server
```

If required, the TigerVNC viewer may also be installed as follows:

```
# dnf install tigervnc
```

Once the server has been installed, the system must be configured to run one or more VNC services and open the appropriate ports on the firewall.

16.4 Assigning Ports to Users

VNC uses a range of ports starting at 5900 to communicate with remote clients. When connecting to a VNC server, these ports are referenced as display numbers (where 5901 is display :1, 5902 is display :2, and so on).

When setting up VNC on AlmaLinux 9, it is helpful to assign a specific port to each remote user to provide consistency in gaining access. Port assignments are declared in the */etc/tigervnc/vncserver.*

users file and use the following format:

```
display_number=user
```

It is recommended that port assignments begin at 5902. For example, the following entry in the *vncserver.users* file assigns display :2 (port 5902) to user *demo*:

```
:2=demo
```

16.5 Configuring the VNC Server

With the VNC server packages installed, the next step is configuring the server. System-wide settings may be declared within the */etc/tigervnc/vncserver-config-defaults* file, while settings for individual users can be placed in the $HOME/.vnc/config file. At a minimum, one of these files should contain the following entries:

```
session=gnome
```

16.6 Setting up a VNC Password

The next step is to specify a password for the remote desktop environment user. While logged in as the remote user, execute the *vncpasswd* command as follows:

```
[demo@demoserver ~]$ vncpasswd
Password:
Verify:
Would you like to enter a view-only password (y/n)? n
A view-only password is not used
```

Next, the firewall needs to be configured to provide external access to the VNC server for remote VNC viewer instances, for example:

```
# firewall-cmd --permanent --add-service=vnc-server
# firewall-cmd --reload
```

16.7 Starting VNC Server

With the service configuration file created, the service needs to be started as follows (where <number> is replaced by the VNC display number:

```
systemctl start vncserver@<number>
```

The following command, for example, starts the VNC server for display :2:

```
# systemctl start vncserver@:2
```

Check that the service has started successfully as follows:

```
# systemctl status vncserver@:2
● vncserver@:2.service - Remote desktop service (VNC)
     Loaded: loaded (/usr/lib/systemd/system/vncserver@.service; enabled; prese>
     Active: active (running) since Thu 2023-08-24 15:50:07 CDT; 21h ago
    Process: 1027 ExecStartPre=/usr/libexec/vncsession-restore :2 (code=exited,>
    Process: 1107 ExecStart=/usr/libexec/vncsession-start :2 (code=exited, stat>
   Main PID: 1114 (vncsession)
      Tasks: 0 (limit: 22131)
```

```
     Memory: 1.9M
        CPU: 38ms
     CGroup: /system.slice/system-vncserver.slice/vncserver@:2.service
             1114 /usr/sbin/vncsession demo :2
```

If the service fails to start, run the *journalctl* command to check for error messages:

```
# journalctl -xe
```

Also, try again after rebooting the system before tying again. If the problem persists, check the VNC log file located in the user's *$HOME/.vnc* directory for diagnostic messages.

16.8 Connecting to a VNC Server

VNC viewer implementations are available for a wide range of operating systems. Therefore, a quick internet search will likely provide numerous links containing details on obtaining and installing this tool on your chosen platform.

First, verify that the remote user has logged out of all local desktop sessions (the VNC server will not start if the user has an active desktop session).

From the desktop of a Linux system on which a VNC viewer such as TigerVNC is installed, a remote desktop connection can be established as follows from a Terminal window:

```
$ vncviewer <hostname>:<display number>
```

In the above example <hostname> is either the hostname or IP address of the remote system, and <display number> is the display number of the VNC server desktop, for example:

```
$ vncviewer 192.168.86.34:2
```

Alternatively, run the command without any options to be prompted for the details of the remote server:

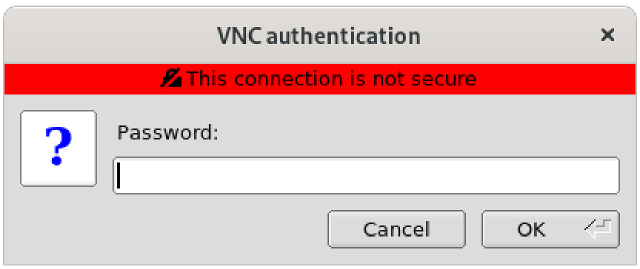

Figure 16-1

Enter the hostname or IP address followed by the display number (for example, 192.168.86.34:2) into the VNC server field and click on the Connect button. The viewer will prompt for the user's VNC password to complete the connection, at which point a new window containing the remote desktop will appear:

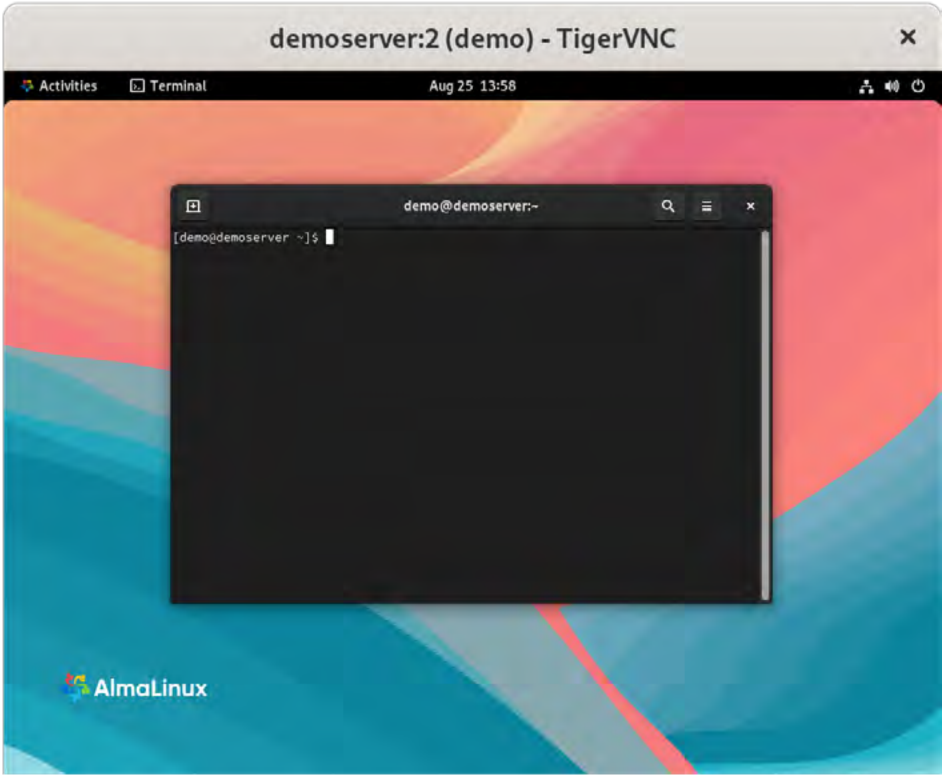

Figure 16-2

This section assumed that the remote desktop was accessed from a Linux or UNIX system; the same steps apply to most other operating systems.

Connecting to a remote VNC server using the steps in this section results in an insecure, unencrypted connection between the client and server. This means the data transmitted during the remote session is vulnerable to interception. Therefore, a few extra steps are necessary to establish a secure and encrypted connection.

16.9 Establishing a Secure Remote Desktop Session

The remote desktop configurations explored in this chapter are considered insecure because no encryption is used. This is acceptable when the remote connection does not extend outside an internal network protected by a firewall. However, a more secure option is needed when a remote session is required over an internet connection. This is achieved by tunneling the remote desktop through a secure shell (SSH) connection. This section will cover how to do this on Linux, UNIX, and macOS client systems.

The SSH server is typically installed and activated by default on AlmaLinux 9 systems. If this is not the case on your system, refer to the chapter *"Configuring SSH Key-based Authentication on AlmaLinux 9"*.

Assuming the SSH server is installed and active, it is time to move to the other system. At the other system, log in to the remote system using the following command, which will establish the secure tunnel between the two systems:

```
$ ssh -l <username> -L 5902:localhost:5902 <remotehost>
```

In the above example, <username> references the user account on the remote system for which VNC access has been configured, and <remotehost> is either the hostname or IP address of the remote system, for example:

```
$ ssh -l neilsmyth -L 5902:localhost:5902 192.168.1.115
```

When prompted, log in using the account password. With the secure connection established, it is time to launch *vncviewer* to use the secure tunnel. Leaving the SSH session running in the other terminal window, launch another terminal and enter the following command:

```
$ vncviewer localhost:5902
```

The *vncviewer* session will prompt for a password if one is required, and then launch the VNC viewer providing secure access to your desktop environment.

Although the connection is now secure and encrypted, the VNC viewer will most likely still report that the connection is insecure. Figure 16-3, for example, shows the warning dialog displayed by the RealVNC viewer running on a macOS system:

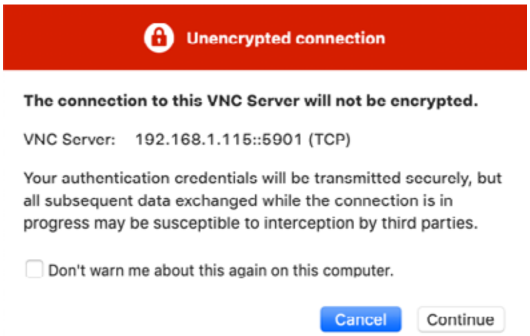

Figure 16-3

Unfortunately, although the connection is now secure, the VNC viewer software has no way of knowing this and consequently continues to issue warnings. However, rest assured that as long as the SSH tunnel is being used, the connection is indeed secure.

In the above example, we left the SSH tunnel session running in a terminal window. If you would prefer to run the session in the background, this can be achieved by using the –f and –N flags when initiating the connection:

```
$ ssh -l <username> -f -N -L 5902:localhost:5902 <remotehost>
```

The above command will prompt for a password for the remote server and then establish the connection in the background, leaving the terminal window available for other tasks.

If you are connecting to the remote desktop from outside the firewall, keep in mind that the IP address for the SSH connection will be the external IP address provided by your ISP or cloud hosting provider, not the LAN IP address of the remote system (since this IP address is not visible to those outside the firewall). Therefore, you will also need to configure your firewall to forward port 22 (for the SSH connection) to the IP address of the system running the desktop. It is not necessary to forward port 5900. Steps to perform port forwarding differ between firewalls, so refer to the documentation for your firewall, router, or wireless base station for details specific to your configuration.

16.10 Establishing a Secure Tunnel on Windows using PuTTY

A similar approach is taken to establishing a secure desktop session from a Windows system to an AlmaLinux 9 server. Assuming you already have a VNC client such as TightVNC installed, the remaining requirement is a Windows SSH client (in this case, PuTTY).

Once PuTTY is downloaded and installed, the first step is establishing a secure connection between the Windows system and the remote AlmaLinux 9 system with appropriate tunneling configured. When launched, PuTTY displays the following screen:

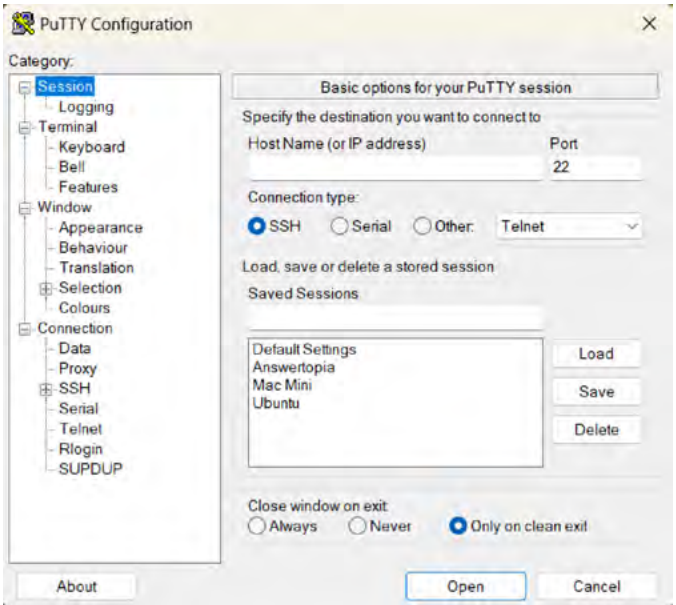

Figure 16-4

Enter the IP address or hostname of the remote host (or the external IP address of the gateway if you are connecting from outside the firewall). The next step is to set up the tunnel. Click on the + next to SSH in the Category tree on the left-hand side of the dialog and select Tunnels. The screen should subsequently appear as follows:

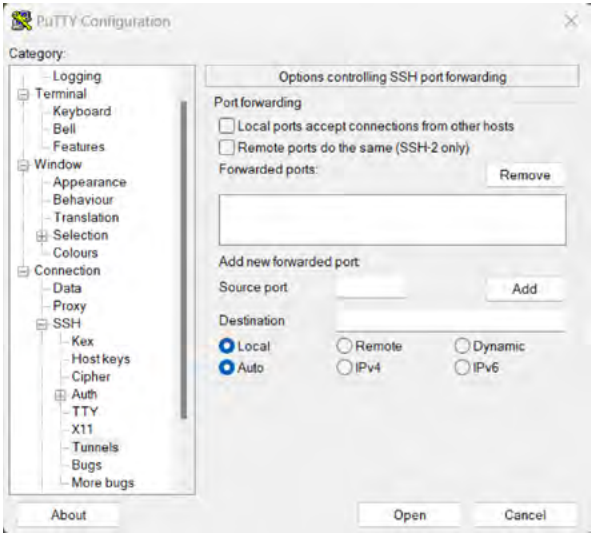

Figure 16-5

Enter 5902 as the Source port and localhost:5902 as the Destination, and click the Add button. Finally, return to the main screen by clicking on the Session category. Enter a name for the session in the Saved Sessions text field and press Save. Click on Open to establish the connection. A terminal window will appear with the login prompt from the remote system. Enter the appropriate user login and password credentials.

The SSH connection is now established. Launch the TightVNC viewer, enter localhost:5902 in the VNC Server text field, and click Connect. The viewer will establish the connection, prompt for the password, and then display the desktop. You are now accessing the remote desktop of a Linux system from Windows over a secure SSH tunnel connection.

16.11 Shutting Down a Desktop Session

To shut down a VNC Server hosted desktop session, use the *systemctl stop* command. For example, to stop desktop :2:

```
# systemctl stop vncserver@:2
```

The VNC server must be stopped before the user attempts to log into a local desktop session. If the user's VNC server is still running, the local desktop session will appear as a blank screen.

16.12 Summary

Remote access to the GNOME desktop environment of an AlmaLinux 9 system can be enabled by using Virtual Network Computing (VNC). Comprising the VNC server running on the remote server and a corresponding client on the local host, VNC allows remote access to multiple desktop instances running on the server.

When the VNC connection is being used over a public connection, SSH tunneling is recommended to ensure that the communication between the client and server is encrypted and secure.

Chapter 20

20. An Overview of Virtualization Techniques

Virtualization is the ability to run multiple operating systems simultaneously on a single computer system. While not necessarily a new concept, Virtualization has come to prominence in recent years because it provides a way to fully utilize the CPU and resource capacity of a server system while providing stability (in that if one virtualized guest system crashes, the host and any other guest systems continue to run).

Virtualization is also helpful in trying out different operating systems without configuring dual boot environments. For example, you can run Windows in a virtual machine without re-partitioning the disk, shut down AlmaLinux 9, and boot from Windows. Instead, you start up a virtualized version of Windows as a guest operating system. Similarly, virtualization allows you to run other Linux distributions within an AlmaLinux 9 system, providing concurrent access to both operating systems.

When deciding on the best approach to implementing virtualization, clearly understanding the different virtualization solutions currently available is essential. Therefore, this chapter's purpose is to describe in general terms the virtualization techniques in common use today.

20.1 Guest Operating System Virtualization

Guest OS virtualization, also called application-based virtualization, is the most straightforward concept to understand. In this scenario, the physical host computer runs a standard unmodified operating system such as Windows, Linux, UNIX, or macOS. Running on this operating system is a virtualization application that executes in much the same way as any other application, such as a word processor or spreadsheet, would run on the system. Within this virtualization application, one or more virtual machines are created to run the guest operating systems on the host computer.

The virtualization application is responsible for starting, stopping, and managing each virtual machine and essentially controlling access to physical hardware resources on behalf of the individual virtual machines. The virtualization application also engages in a process known as binary rewriting, which involves scanning the instruction stream of the executing guest system and replacing any privileged instructions with safe emulations. This makes the guest system think it is running directly on the system hardware rather than in a virtual machine within an application.

The following figure illustrates guest OS-based virtualization:

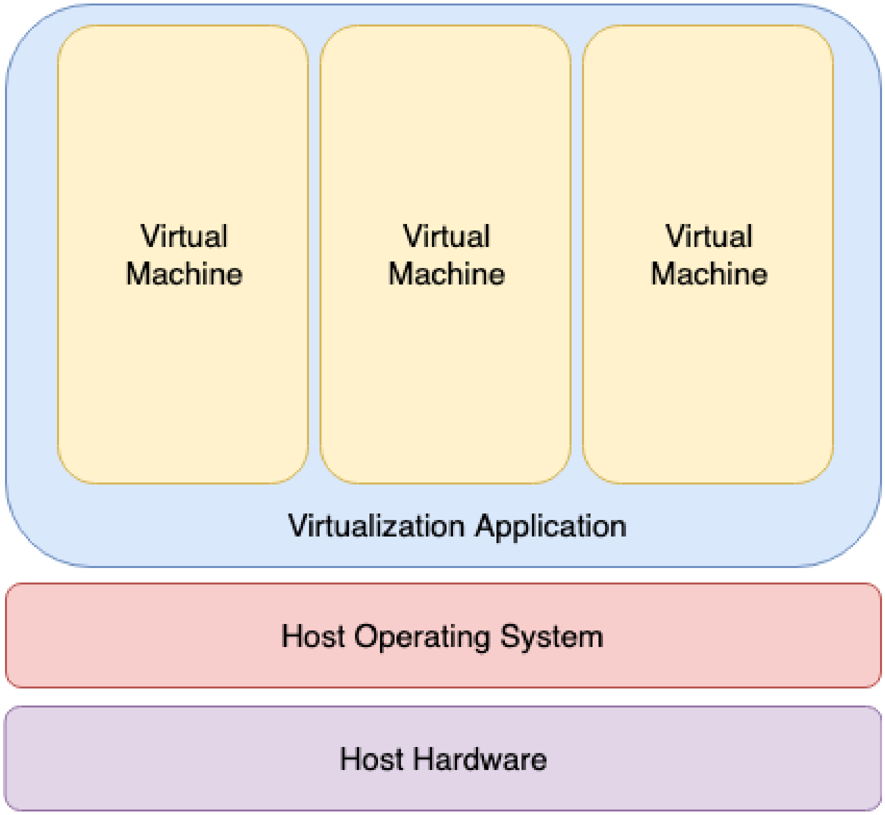

Figure 20-1

As outlined in the above diagram, the guest operating systems operate in virtual machines within the virtualization application, which, in turn, runs on top of the host operating system in the same way as any other application. The multiple layers of abstraction between the guest operating systems and the underlying host hardware are not conducive to high levels of virtual machine performance. However, this technique has the advantage that no changes are necessary to host or guest operating systems, and no special CPU hardware virtualization support is required.

20.2 Hypervisor Virtualization

In hypervisor virtualization, the task of a hypervisor is to handle resource and memory allocation for the virtual machines and provide interfaces for higher-level administration and monitoring tools. Hypervisor-based solutions are categorized as being either Type-1 or Type-2.

Type-2 hypervisors (sometimes called hosted hypervisors) are installed as software applications that run on top of the host operating system, providing virtualization capabilities by coordinating access to resources such as the CPU, memory, and network for guest virtual machines. Figure 21-2 illustrates the typical architecture of a system using Type-2 hypervisor virtualization:

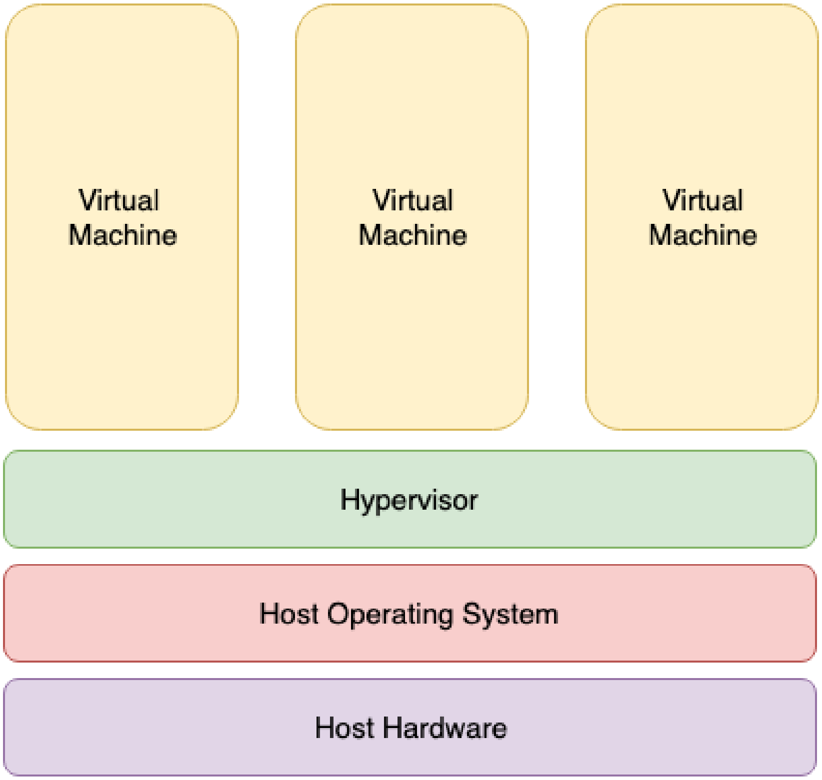

Figure 20-2

To understand how Type-1 hypervisors work, it helps to understand Intel x86 processor architecture. The x86 family of CPUs provides a range of protection levels known as rings in which code can execute. Ring 0 has the highest level privilege, and it is in this ring that the operating system kernel normally runs. Code executing in ring 0 is said to be running in system space, kernel mode, or supervisor mode. All other code, such as applications running on the operating system, operate in less privileged rings, typically ring 3.

In contrast to Type-2 hypervisors, Type-1 hypervisors (also referred to as metal or native hypervisors) run directly on the hardware of the host system in ring 0. With the hypervisor occupying ring 0 of the CPU, the kernels for any guest operating systems running on the system must run in less privileged CPU rings. Unfortunately, most operating system kernels are written explicitly to run in ring 0 because they need to perform tasks only available in that ring, such as the ability to execute privileged CPU instructions and directly manipulate memory. Several different solutions to this problem have been devised in recent years, each of which is described below:

20.2.1 Paravirtualization

Under paravirtualization, the kernel of the guest operating system is modified specifically to run on the hypervisor. This typically involves replacing privileged operations that only run in ring 0 of the CPU with calls to the hypervisor (known as hypercalls). The hypervisor, in turn, performs the task on behalf of the guest kernel. Unfortunately, this typically limits support to open-source operating systems such as Linux, which may be freely altered, and proprietary operating systems where the owners have agreed to make the necessary code modifications to target a specific hypervisor. These issues notwithstanding, the ability of the guest kernel to communicate directly with the hypervisor results in greater performance levels than other virtualization approaches.

20.2.2 Full Virtualization

Full virtualization provides support for unmodified guest operating systems. The term unmodified refers to operating system kernels that have not been altered to run on a hypervisor and, therefore, still execute privileged operations as though running in ring 0 of the CPU. In this scenario, the hypervisor provides CPU emulation to handle and modify privileged and protected CPU operations made by unmodified guest operating system kernels. Unfortunately, this emulation process requires both time and system resources to operate, resulting in inferior performance levels when compared to those provided by paravirtualization.

20.2.3 Hardware Virtualization

Hardware virtualization leverages virtualization features built into the latest generations of CPUs from both Intel and AMD. These technologies, called Intel VT and AMD-V, respectively, provide extensions necessary to run unmodified guest virtual machines without the overheads inherent in full virtualization CPU emulation. In very simplistic terms, these processors provide an additional privilege mode (ring -1) above ring 0 in which the hypervisor can operate, thereby leaving ring 0 available for unmodified guest operating systems.

The following figure illustrates the Type-1 hypervisor approach to virtualization:

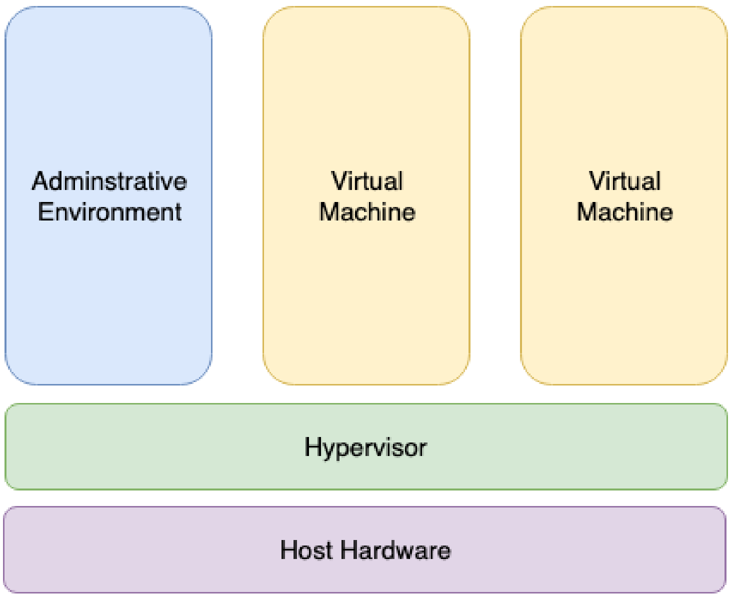

Figure 20-3

As outlined in the above illustration, in addition to the virtual machines, an administrative operating system or management console also runs on top of the hypervisor allowing the virtual machines to be managed by a system administrator.

20.3 Virtual Machine Networking

Virtual machines will invariably need to be connected to a network to be of any practical use. One option is for the guest to be connected to a virtual network running within the host computer's operating system. In this configuration, any virtual machines on the virtual network can see each other, but Network Address Translation (NAT) provides access to the external network. When using the virtual network and NAT, each virtual machine is represented on the external network (the network to which the host is connected) using the IP address of the host system. This is the default behavior for KVM virtualization on AlmaLinux 9 and generally requires no additional configuration. Typically, a single virtual network is created by default, represented by the name *default* and the device *virbr0*.

For guests to appear as individual and independent systems on the external network (i.e., with their own IP addresses), they must be configured to share a physical network interface on the host. The quickest way to achieve this is to configure the virtual machine to use the "direct connection" network configuration option (also called MacVTap), which will provide the guest system with an IP address on the same network as the host. Unfortunately, while this gives the virtual machine access to other systems on the network, it is not possible to establish a connection between the guest and the host when using the MacVTap driver.

A better option is to configure a network bridge interface on the host system to which the guests can connect. This provides the guest with an IP address on the external network while also allowing the guest and host to communicate, a topic covered in the chapter entitled *"Creating an AlmaLinux 9 KVM Networked Bridge Interface"*.

20.4 Summary

Virtualization is the ability to run multiple guest operating systems within a single host operating system. Several approaches to virtualization have been developed, including a guest operating system and hypervisor virtualization. Hypervisor virtualization falls into two categories known as Type-1 and Type-2. Type-2 virtualization solutions are categorized as paravirtualization, full virtualization, and hardware virtualization, the latter using special virtualization features of some Intel and AMD processor models.

Virtual machine guest operating systems have several options in terms of networking, including NAT, direct connection (MacVTap), and network bridge configurations.

Chapter 25

25. Creating an AlmaLinux 9 KVM Networked Bridge Interface

By default, the KVM virtualization environment on AlmaLinux 9 creates a virtual network to which virtual machines may connect. It is also possible to configure a direct connection using a MacVTap driver. However, as outlined in the chapter entitled *"An Overview of Virtualization Techniques"*, this approach does not allow the host and guest systems to communicate.

This chapter will cover the steps involved in creating a network bridge on AlmaLinux 9, enabling guest systems to share one or more of the host system's physical network connections while still allowing the guest and host systems to communicate.

In the remainder of this chapter, we will explain how to configure an AlmaLinux 9 network bridge for KVM-based guest operating systems.

25.1 Getting the Current Network Manager Settings

A network bridge can be created using the NetworkManager command-line interface tool (nmcli). The NetworkManager is installed and enabled by default on AlmaLinux 9 systems and is responsible for detecting and connecting to network devices and providing an interface for managing networking configurations.

A list of current network connections on the host system can be displayed as follows:

```
# nmcli con show
NAME    UUID                                  TYPE      DEVICE
eno1    f8d7c1b3-994c-3b5a-87f1-f8430e49f992  ethernet  eno1
lo      05d13946-2ae4-49b7-92b5-1ecfbe604adb  loopback  lo
virbr0  91d7e0e0-d953-446a-887b-62c2635951e2  bridge    virbr0
```

The above output shows that the host has an Ethernet network connection established via a device named eno1 and the default bridge interface named virbr0, which provides access to the NAT-based virtual network to which KVM guest systems are connected by default.

Similarly, the following command can be used to identify the devices (both virtual and physical) that are currently configured on the system:

```
# nmcli device show
GENERAL.DEVICE:                         eno1
GENERAL.TYPE:                           ethernet
GENERAL.HWADDR:                         00:23:24:52:52:57
GENERAL.MTU:                            1500
GENERAL.STATE:                          100 (connected)
GENERAL.CONNECTION:                     eno1
```

```
GENERAL.CON-PATH:                       /org/freedesktop/NetworkManager/
ActiveConnection/2
WIRED-PROPERTIES.CARRIER:               on
IP4.ADDRESS[1]:                         192.168.86.39/24
IP4.GATEWAY:                            192.168.86.1
IP4.ROUTE[1]:                           dst = 192.168.86.0/24, nh = 0.0.0.0, mt =
100
IP4.ROUTE[2]:                           dst = 0.0.0.0/0, nh = 192.168.86.1, mt =
100
IP4.DNS[1]:                             192.168.86.1
IP4.DOMAIN[1]:                          lan
IP6.ADDRESS[1]:                         fd1e:fe64:8988:2c34:223:24ff:fe52:5257/64
IP6.ADDRESS[2]:                         fe80::223:24ff:fe52:5257/64
.
.
```

The above partial output indicates that the host system on which the command was executed contains a physical Ethernet device (eno1) and a virtual bridge (virbr0).

The virsh command may also be used to list the virtual networks currently configured on the system:

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

Currently, the only virtual network present is the default network provided by virbr0. Now that some basic information about the current network configuration has been obtained, the next step is to create a network bridge connected to the physical network device (in this case, eno1).

25.2 Creating a Network Manager Bridge from the Command-Line

The first step in creating the network bridge is adding a new connection to the configuration. This can be achieved using the *nmcli* tool, specifying that the connection is to be a bridge and providing names for both the connection and the interface:

```
# nmcli con add ifname br0 type bridge con-name br0
```

Once the connection has been added, a bridge slave interface needs to be established between physical device eno1 (the slave) and the bridge connection br0 (the master) as follows:

```
# nmcli con add type bridge-slave ifname eno1 master br0
Connection 'bridge-slave-eno1' (07e588c0-14a1-4168-a9c6-e9056f55c11f)
successfully added.
```

At this point, the NetworkManager connection list should read as follows:

```
# nmcli con show
eno1              f8d7c1b3-994c-3b5a-87f1-f8430e49f992  ethernet  eno1
virbr0            91d7e0e0-d953-446a-887b-62c2635951e2  bridge    virbr0
br0               5eac7f23-d0fa-4df9-986b-b643f1b4d35b  bridge    --
```

```
bridge-slave-eno1  07e588c0-14a1-4168-a9c6-e9056f55c11f  ethernet  --
```

The next step is to start up the bridge interface. If the steps to configure the bridge are being performed over a network connection (i.e., via SSH) this step can be problematic because the current eno1 connection must be closed down before the bridge connection can be brought up. This means the current connection will be lost before the bridge connection can be enabled to replace it, potentially leaving the remote host unreachable.

If you are accessing the host system remotely, this problem can be avoided by creating a shell script to perform the network changes. This will ensure that the bridge interface is enabled after the eno1 interface is brought down, allowing you to reconnect to the host after the changes are complete. Begin by creating a shell script file named *bridge.sh* containing the following commands:

```
#!/bin/bash
nmcli con down eno1
nmcli con up br0
```

Once the script has been created, execute it as follows:

```
# sh ./bridge.sh
```

When the script executes, the connection will be lost when the eno1 connection is brought down. After waiting a few seconds, however, it should be possible to reconnect to the host once the br0 connection has been activated. Note that in some cases, the bridge interface may be assigned a different IP address than the one previously assigned to the system. Keep this in mind while attempting to reconnect via ssh.

If you are working locally on the host, the two *nmcli* commands can be run within a terminal window without any risk of losing connectivity:

```
# nmcli con down eno1
# nmcli con up br0
```

Once the bridge is up and running, the connection list should now include both the bridge and the bridge-slave connections:

```
# nmcli con show
NAME               UUID                                  TYPE      DEVICE
br0                5eac7f23-d0fa-4df9-986b-b643f1b4d35b  bridge    br0
virbr0             91d7e0e0-d953-446a-887b-62c2635951e2  bridge    virbr0
bridge-slave-eno1  07e588c0-14a1-4168-a9c6-e9056f55c11f  ethernet  eno1
eno1               f8d7c1b3-994c-3b5a-87f1-f8430e49f992  ethernet  --
```

Note that the connection is still listed but is no longer active. To exclude inactive connections from the list, use the --active flag when requesting the list:

```
# nmcli con show --active
NAME               UUID                                  TYPE      DEVICE
br0                5eac7f23-d0fa-4df9-986b-b643f1b4d35b  bridge    br0
virbr0             91d7e0e0-d953-446a-887b-62c2635951e2  bridge    virbr0
bridge-slave-eno1  07e588c0-14a1-4168-a9c6-e9056f55c11f  ethernet  eno1
```

25.3 Declaring the KVM Bridged Network

At this point, the bridge connection is on the system but is not visible to the KVM environment. Running the virsh command should still list the default network as being the only available network option:

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

Before a virtual machine can use the bridge, it must be declared and added to the KVM network configuration. This involves the creation of a definition file and, once again, using the *virsh* command-line tool.

Begin by creating a definition file for the bridge network named *bridge.xml* that reads as follows:

```
<network>
  <name>br0</name>
  <forward mode="bridge"/>
  <bridge name="br0" />
</network>
```

Next, use the file to define the new network:

```
# virsh net-define ./bridge.xml
Network br0 defined from ./bridge.xml
```

Once the network has been defined, start it and, if required, configure it to autostart each time the system reboots:

```
# virsh net-start br0
# virsh net-autostart br0
```

Once again, list the networks to verify that the bridge network is now accessible within the KVM environment:

```
# virsh net-list --all
 Name                 State      Autostart     Persistent
----------------------------------------------------------
 br0                  active     yes           yes
 default              active     yes           yes
```

25.4 Using a Bridge Network in a Virtual Machine

To create a virtual machine that uses the bridge network, use the *virt-install --network* option and specify the br0 bridge name. For example:

```
# virt-install --name alma_vm_guest --memory 1024 --disk path=/tmp/alma_vm_guest.
img,size=10 --network network=br0 --cdrom /home/demo/AlmaLinux-9.2-x86_64-
minimal.iso
```

When the guest operating system runs, it will appear on the same physical network as the host system and will no longer be on the NAT-based virtual network.

The bridge may also be selected for virtual machines within the Cockpit interface by editing the virtual machine, locating the Network interfaces section, and clicking the Edit button as highlighted in Figure 25-1 below:

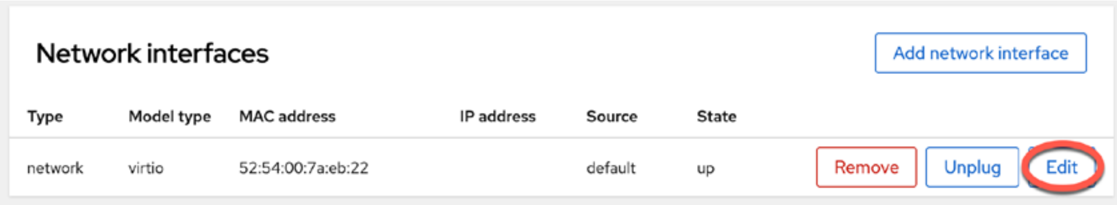

Figure 25-1

Within the resulting interface settings dialog, change the Interface type menu to Bridge to LAN and set the Source to br0 as shown in Figure 25-2:

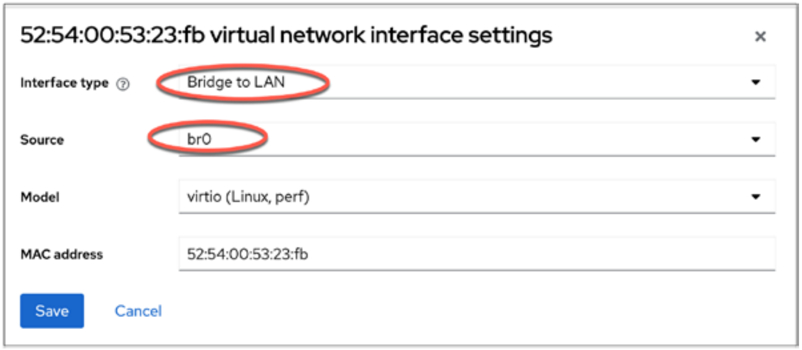

Figure 25-2

Similarly, when creating a new virtual machine using the *virt-manager* tool, the bridge will be available within the Network selection menu:

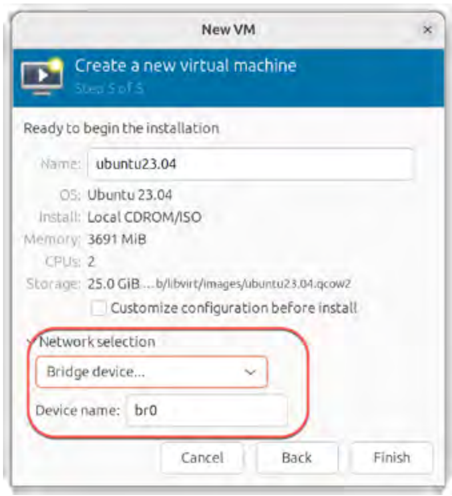

Figure 25-3

To modify an existing virtual machine so that it uses the bridge, use the *virsh edit* command. This command loads the XML definition file into an editor where changes can be made and saved:

```
# virsh edit GuestName
```

By default, the file will be loaded into the vi editor. To use a different editor, change the $EDITOR environment variable, for example:

```
# export EDITOR=gedit
```

To change from the default virtual network, locate the <interface> section of the file, which will read as follows for a NAT-based configuration:

```
<interface type='network'>
      <mac address='<your mac address here>'/>
      <source network='default'/>
      <model type='virtio'/>
      <address type='pci' domain='0x0000' bus='0x01' slot='0x00' function='0x0'/>
</interface>
```

Alternatively, if the virtual machine was using a direct connection, the entry may read as follows:

```
<interface type='direct'>
      <mac address='<your mac address here>'/>
      <source dev='eno1' mode='vepa'/>
      <model type='virtio'/>
      <address type='pci' domain='0x0000' bus='0x01' slot='0x00' function='0x0'/>
```

To use the bridge, change the source network property to read as follows before saving the file:

```
<interface type='network'>
      <mac address='<your mac address here>'/>
      <source network='br0'/>
      <model type='virtio'/>
      <address type='pci' domain='0x0000' bus='0x01' slot='0x00' function='0x0'/>
</interface>
```

If the virtual machine is already running, the change will not take effect until it is restarted.

25.5 Creating a Bridge Network using nm-connection-editor

If either local or remote desktop access is available on the host system, much of the bridge configuration process can be performed using the *nm-connection-editor* graphical tool. To use this tool, open a Terminal window within the desktop and enter the following command:

```
# nm-connection-editor
```

When the tool has loaded, the window shown in Figure 25-4 will appear, listing the currently configured network connections (essentially the same output as that generated by the *nmcli con show* command):

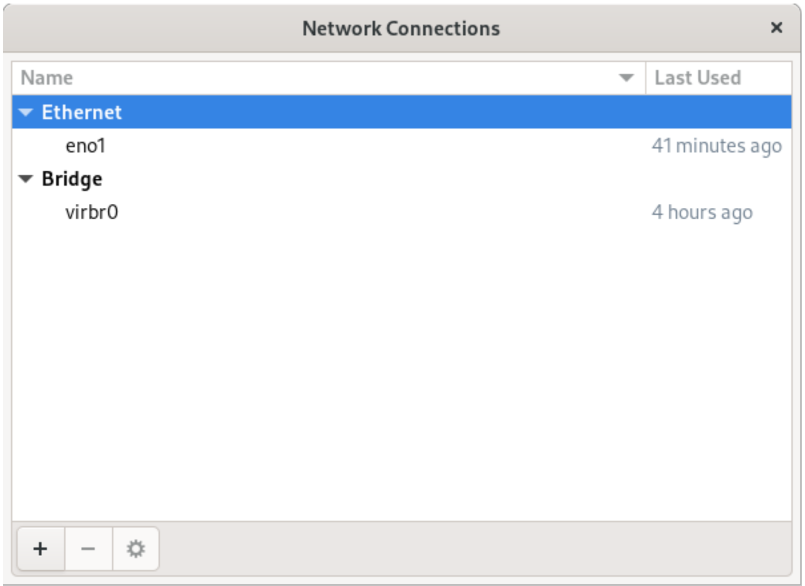

Figure 25-4

To create a new connection, click on the '+' button in the window's bottom left-hand corner. Then, from the resulting dialog (Figure 25-5), select the Bridge option from the menu:

Figure 25-5

With the bridge option selected, click the Create button to proceed to the bridge configuration screen. Begin by changing both the connection and interface name fields to br0 before clicking on the Add button located to the right of the Bridge connections list, as highlighted in Figure 25-6:

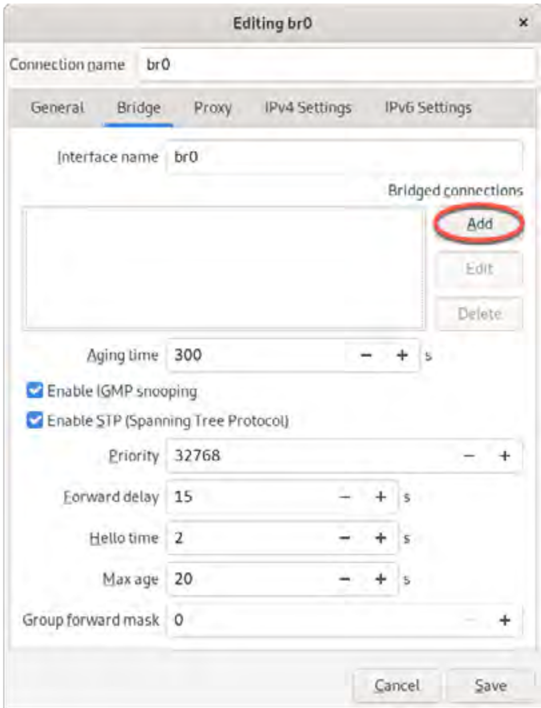

Figure 25-6

From the connection type dialog (Figure 25-7), change the menu setting to Ethernet before clicking on the Create button:

Figure 25-7

Another dialog will now appear in which the bridge slave connection needs to be configured. Within this dialog, select the physical network to which the bridge is to connect (for example, eno1) from the Device menu:

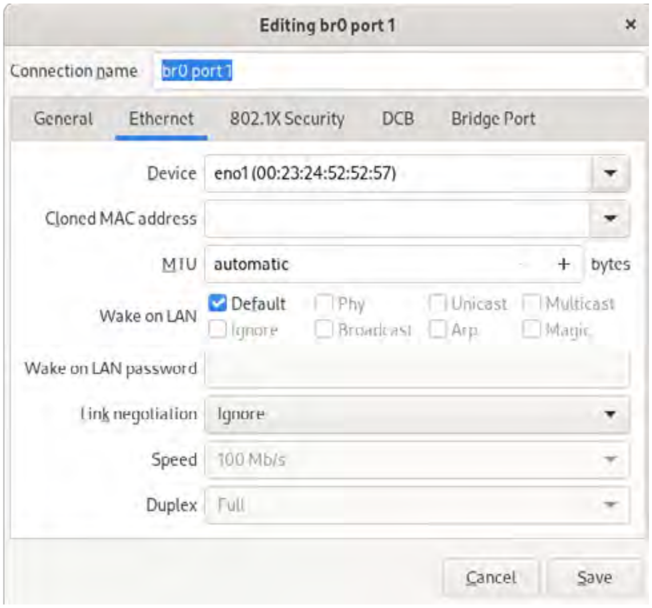

Figure 25-8

Click on the Save button to apply the changes and return to the Editing br0 dialog (as illustrated in Figure 25-6 above). Within this dialog, click on the Save button to create the bridge. On returning to the main window, the new bridge and slave connections should now be listed:

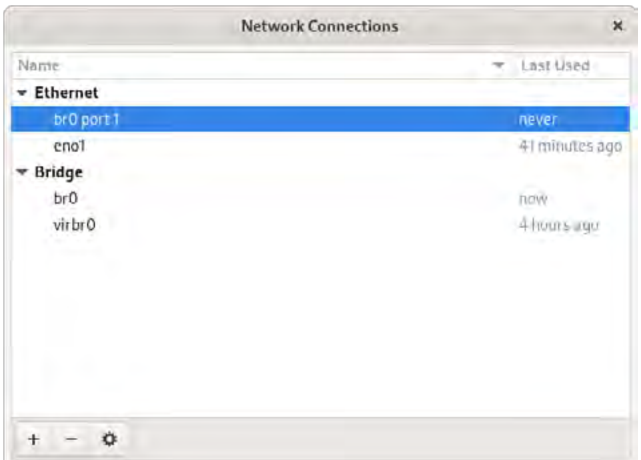

Figure 25-9

All that remains is to bring down the original eno1 connection and bring up the br0 connection using the steps outlined in the previous chapter (remembering to perform these steps in a shell script if the host is being accessed remotely):

```
# nmcli con down eno1
# nmcli con up br0
```

It will also be necessary, as it was when creating the bridge using the command-line tool, to add this bridge to the KVM network configuration. To do so, repeat the steps outlined in the *"Declaring the KVM Bridged Network"* section above. Once this step has been taken, the bridge is ready to be used by guest virtual machines.

25.6 Summary

By default, KVM virtual machines are connected to a virtual network that uses NAT to provide access to the network to which the host system is connected. If the guests are required to appear on the network with their own IP addresses, they need to be configured to share the physical network interface of the host system. This chapter outlines that this can be achieved using the *nmcli* or *nm-connection-editor* tools to create a networked bridge interface.

Chapter 27

27. An Introduction to Linux Containers

The preceding chapters covered the concept of virtualization, emphasizing creating and managing virtual machines using KVM. This chapter will introduce a related technology in the form of Linux Containers. While there are some similarities between virtual machines and containers, key differences will be outlined in this chapter, along with an introduction to the concepts and advantages of Linux Containers. The chapter will also introduce some AlmaLinux 9 container management tools. Once the basics of containers have been covered in this chapter, the next chapter will work through some practical examples of creating and running containers on AlmaLinux 9.

27.1 Linux Containers and Kernel Sharing

In simple terms, Linux containers are a lightweight alternative to virtualization. A virtual machine contains and runs the entire guest operating system in a virtualized environment. The virtual machine, in turn, runs on top of an environment such as a hypervisor that manages access to the physical resources of the host system.

Containers work by using a concept referred to as kernel sharing, which takes advantage of the architectural design of Linux and UNIX-based operating systems.

To understand how kernel sharing and containers work, it helps first to understand the two main components of Linux or UNIX operating systems. At the core of the operating system is the kernel. In simple terms, the kernel handles all the interactions between the operating system and the physical hardware. The second key component is the root file system which contains all the libraries, files, and utilities necessary for the operating system to function. Taking advantage of this structure, containers each have their own root file system but share the host operating system's kernel. This structure is illustrated in the architectural diagram in Figure 27-1 below.

This type of resource sharing is made possible by the ability of the kernel to dynamically change the current root file system (a concept known as change root or chroot) to a different root file system without having to reboot the entire system. Linux containers are essentially an extension of this capability combined with a container runtime, the responsibility of which is to provide an interface for executing and managing the containers on the host system. Several container runtimes are available, including Docker, lxd, containerd, and CRI-O.

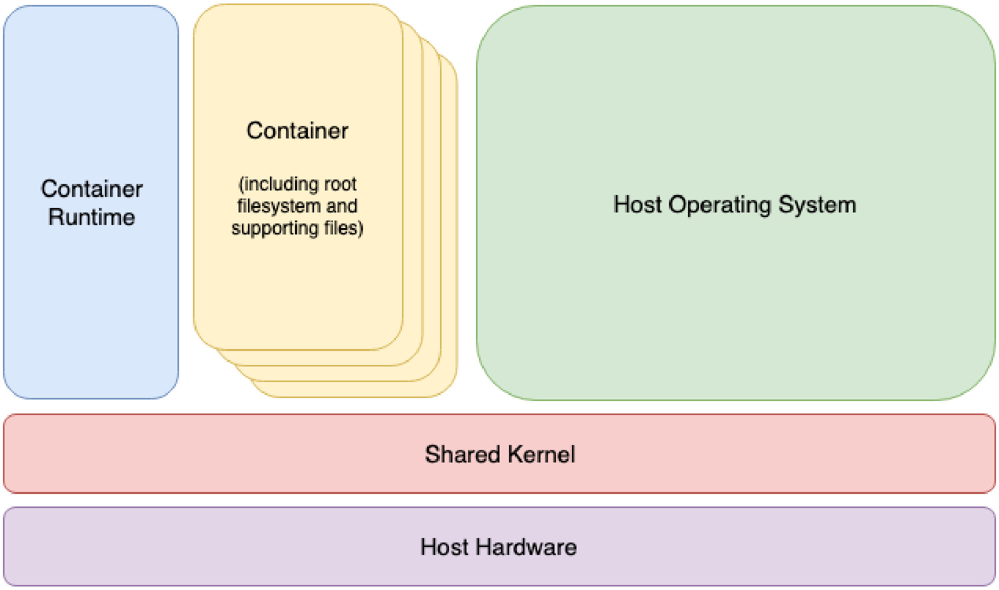

Figure 27-1

27.2 Container Uses and Advantages

The main advantage of containers is that they require considerably less resource overhead than virtualization allowing many container instances to be run simultaneously on a single server. They can be started and stopped rapidly and efficiently in response to demand levels. In addition, containers run natively on the host system providing a level of performance that a virtual machine cannot match.

Containers are also highly portable and can be easily migrated between systems. Combined with a container management system such as Docker, OpenShift, and Kubernetes, it is possible to deploy and manage containers on a vast scale spanning multiple servers and cloud platforms, potentially running thousands of containers.

Containers are frequently used to create lightweight execution environments for applications. In this scenario, each container provides an isolated environment containing the application together with all of the runtime and supporting files required by that application to run. The container can then be deployed to any other compatible host system that supports container execution and runs without any concerns that the target system may not have the necessary runtime configuration for the application - all of the application's dependencies are already in the container.

Containers are also helpful when bridging the gap between development and production environments. By performing development and QA work in containers, they can be passed to production and launched safely because the applications run in the same container environments in which they were developed and tested.

Containers also promote a modular approach to deploying large and complex solutions. Instead of developing applications as single monolithic entities, containers can be used to design applications as groups of interacting modules, each running in a separate container.

One possible drawback of containers is that the guest operating systems must be compatible with the shared kernel version. It is not, for example, possible to run Microsoft Windows in a container on a Linux system. Nor is it possible for a Linux guest system designed for the 2.6 version of the kernel to share a 2.4 version kernel. These requirements are not, however, what containers were designed for. Rather than being seen as limitations, these restrictions should be considered some of the key advantages of containers in providing a simple, scalable, and reliable deployment platform.

27.3 AlmaLinux 9 Container Tools

AlmaLinux 9 provides several tools for creating, inspecting, and managing containers. The main tools are as follows:

- **buildah** – A command-line tool for building container images.
- **podman** – A command-line based container runtime and management tool. Performs tasks such as downloading container images from remote registries and inspecting, starting, and stopping images.
- **skopeo** – A command-line utility used to convert container images, copy images between registries and inspect images stored in registries without downloading them.
- **runc** – A lightweight container runtime for launching and running containers from the command line.
- **OpenShift** – An enterprise-level container application management platform consisting of command-line and web-based tools.

All of the above tools comply with the Open Container Initiative (OCI), a set of specifications designed to ensure that containers conform to the same standards between competing tools and platforms.

27.4 The Docker Registry

Although AlmaLinux 9 is provided with a set of tools designed to be used in place of those provided by Docker, those tools still need access to AlmaLinux images for use when building containers. For this purpose, the AlmaLinux OS Foundation maintains a set of container images within the Docker Hub. The Docker Hub is an online container registry made of multiple repositories, each containing a wide range of container images available for download when building containers. The images within a repository are each assigned a repository tag (for example, 9.2, latest, etc.) which can be referenced when performing an image download. The following, for example, is the URL of the latest AlmaLinux image contained within the Docker Hub:

```
docker://docker.io/library/almalinux
```

In addition to downloading (referred to as "pulling" in container terminology) container images from Docker and other third-party hosts registries, you can also use registries to store your own images. This can be achieved either by hosting your own registry, or by making use of existing services such as those provided by Docker, Amazon AWS, Google Cloud, Microsoft Azure, and IBM Cloud, to name a few of the many options.

27.5 Container Networking

By default, containers are connected to a network using a Container Networking Interface (CNI) bridged network stack. In the bridged configuration, all the containers running on a server belong to the same subnet and, as such, can communicate with each other. The containers are also connected to the external network by bridging the host system's network connection. Similarly, the host can access the containers via a virtual network interface (usually named podman0) which will have been created as part of the container tool installation.

27.6 Summary

Linux Containers offer a lightweight alternative to virtualization and take advantage of the structure of the Linux and Unix operating systems. Linux Containers share the host operating system's kernel, with each container having its own root file system containing the files, libraries, and applications. As a result, containers are highly efficient and scalable and provide an ideal platform for building and deploying modular enterprise-level solutions. In addition, several tools and platforms are available for building, deploying, and managing containers, including third-party solutions and those provided by the AlmaLinux OS Foundation.

Chapter 32

32. Adding a New Disk to an AlmaLinux 9 Volume Group and Logical Volume

In the previous chapter, we looked at adding a new disk drive to an AlmaLinux 9 system, creating a partition and file system, and then mounting that file system to access the disk. An alternative to creating fixed partitions and file systems is to use Logical Volume Management (LVM) to create logical disks comprising space from one or more physical or virtual disks or partitions. The advantage of using LVM is that space can be added to or removed from logical volumes without spreading data over multiple file systems.

Let us take, for example, the file system of an AlmaLinux 9-based server. Without LVM, this file system would be created with a specific size when the operating system is installed. If a new disk drive is installed, there is no way to allocate any of that space to the / file system. The only option would be to create new file systems on the new disk and mount them at particular mount points. In this scenario, you would have plenty of space on the new file system, but the / file system would still be nearly full. The only option would be to move files onto the new file system. With LVM, the new disk (or part thereof) can be assigned to the logical volume containing the home file system, thereby dynamically extending the space available.

In this chapter, we will look at the steps necessary to add new disk space to both a volume group and a logical volume to add additional space to the home file system of an AlmaLinux 9 system.

32.1 An Overview of Logical Volume Management (LVM)

LVM provides a flexible and high-level approach to managing disk space. Instead of each disk drive being split into partitions of fixed sizes onto which fixed-size file systems are created, LVM provides a way to group disk space into logical volumes that can be easily resized and moved. In addition, LVM allows administrators to carefully control disk space assigned to different groups of users by allocating distinct volume groups or logical volumes to those users. When the space initially allocated to the volume is exhausted, the administrator can add more space without moving the user files to a different file system.

LVM consists of the following components:

32.1.1 Volume Group (VG)

The Volume Group is the high-level container with one or more logical and physical volumes.

32.1.2 Physical Volume (PV)

A physical volume represents a storage device such as a disk drive or other storage media.

32.1.3 Logical Volume (LV)

A logical volume is equivalent to a disk partition and, as with a disk partition, can contain a file system.

32.1.4 Physical Extent (PE)

Each physical volume (PV) is divided into equal-sized blocks known as physical extents.

32.1.5 Logical Extent (LE)

Each logical volume (LV) is divided into equal size blocks called logical extents.

Suppose we are creating a new volume group called VolGroup001. This volume group needs physical disk space to function, so we allocate three disk partitions */dev/sda1*, */dev/sdb1*, and */dev/sdb2*. These become physical volumes in VolGroup001. We would then create a logical volume called LogVol001 within the volume group comprising the three physical volumes.

If we run out of space in LogVol001, we add more disk partitions as physical volumes and assign them to the volume group and logical volume.

32.2 Getting Information about Logical Volumes

As an example of using LVM with AlmaLinux 9, we will work through an example of adding space to the / file system of a standard AlmaLinux 9 installation. Anticipating the need for flexibility in the sizing of the partition, AlmaLinux 9 sets up the / file system as a logical volume (called) within a volume group called *almalinux*. Before making any changes to the LVM setup, however, it is essential first to gather information.

Running the *mount* command will output information about a range of mount points, including the following entry for the home filesystem:

```
/dev/mapper/almalinux-home on /home type xfs (rw,relatime,seclabel,attr2,inode64,
logbufs=8,logbsize=32k,noquota)
```

Information about the volume group can be obtained using the *vgdisplay* command:

```
# vgdisplay
  --- Volume group ---
  VG Name               almalinux
  System ID
  Format                lvm2
  Metadata Areas        1
  Metadata Sequence No  4
  VG Access             read/write
  VG Status             resizable
  MAX LV                0
  Cur LV                3
```

```
  Open LV               3
  Max PV                0
  Cur PV                1
  Act PV                1
  VG Size               296.50 GiB
  PE Size               4.00 MiB
  Total PE              75904
  Alloc PE / Size       75904 / 296.50 GiB
  Free  PE / Size       0 / 0
  VG UUID               HSp6WF-NrHn-KHrv-NbI8-jDhe-WTpc-Lb1CNa
```

As we can see in the above example, the *almalinux* volume group has a physical extent size of 4.00MiB and has a total of 296.50GB available for allocation to logical volumes. Currently, 75904 physical extents are allocated, equaling the total capacity. Therefore, we must add one or more physical volumes to increase the space allocated to any logical volumes in the *almalinux* volume group. The *vgs* tool is also helpful for displaying a quick overview of the space available in the volume groups on a system:

```
# vgs
  VG        #PV #LV #SN Attr   VSize   VFree
  almalinux   1   3   0 wz--n- 296.50g    0
```

Information about logical volumes in a volume group may similarly be obtained using the *lvdisplay* command:

```
# lvdisplay
  --- Logical volume ---
  LV Path                /dev/almalinux/swap
  LV Name                swap
  VG Name                almalinux
  LV UUID                GwyCy4-JjCg-Nj1l-cmWf-GttL-MHwJ-YmaDYV
  LV Write Access        read/write
  LV Creation host, time demoserver, 2023-08-17 15:48:07 -0500
  LV Status              available
  # open                 2
  LV Size                3.75 GiB
  Current LE             961
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           253:1

  --- Logical volume ---
  LV Path                /dev/almalinux/home
  LV Name                home
  VG Name                almalinux
```

```
  LV UUID                lFAhky-CV0Z-Wc4Z-fqco-dGmM-10dk-veFJj9
  LV Write Access        read/write
  LV Creation host, time demoserver, 2023-08-17 15:48:07 -0500
  LV Status              available
  # open                 1
  LV Size                <222.75 GiB
  Current LE             57023
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           253:2

  --- Logical volume ---
  LV Path                /dev/almalinux/root
  LV Name                root
  VG Name                almalinux
  LV UUID                rGk5UZ-X0sJ-Lb3x-Lhe8-je8e-EWoo-609AfW
  LV Write Access        read/write
  LV Creation host, time demoserver, 2023-08-17 15:48:09 -0500
  LV Status              available
  # open                 1
  LV Size                70.00 GiB
  Current LE             17920
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           253:0
```

As shown in the above example, 70 GiB of the space in volume group *almalinux* is allocated to logical volume *root* (for the / file system), approximately 222 GiB to the home volume group (for */home*), and 3.75 GiB to *swap* (for swap space).

Now that we know what space is being used, it is often helpful to understand which devices are providing the space (in other words, which devices are being used as physical volumes). To obtain this information, we need to run the *pvdisplay* command:

```
# pvdisplay
  --- Physical volume ---
  PV Name               /dev/sda2
  VG Name               almalinux
  PV Size               296.50 GiB / not usable 4.00 MiB
  Allocatable           yes (but full)
  PE Size               4.00 MiB
  Total PE              75904
```

```
  Free PE               0
  Allocated PE          75904
  PV UUID               GboISU-O0WH-fdEU-3sre-mHr0-T1X9-ObypcW
```

Clearly, the space controlled by logical volume *almalinux* is provided via a physical volume located on */dev/sda2*.

Now that we know more about our LVM configuration, we can add space to the volume group and the logical volume contained within.

32.3 Adding Additional Space to a Volume Group from the Command Line

Just as with the previous steps to gather information about the current Logical Volume Management configuration of an AlmaLinux 9 system, changes to this configuration can be made from the command line.

In the remainder of this chapter, we will assume that a new disk has been added to the system and that the operating system sees it as */dev/sdb*. We shall also assume this is a new disk with no existing partitions. If existing partitions are present, they should be backed up, and then the partitions should be deleted from the disk using the *fdisk* utility. For example, assuming a device represented by */dev/sdb* containing two partitions as follows:

```
# fdisk -l /dev/sdb
Disk /dev/sdb: 14.46 GiB, 15525216256 bytes, 30322688 sectors
Disk model: USB 2.0 FD
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x4c33060b

Device     Boot Start      End  Sectors  Size Id Type
/dev/sdb1        2048 30322687 30320640 14.5G 83 Linux
```

Once any filesystems on these partitions have been unmounted, they can be deleted as follows:

```
# fdisk /dev/sdb

Welcome to fdisk (util-linux 2.37.4).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Command (m for help): d
Selected partition 1
Partition 1 has been deleted.

Command (m for help): w
```

```
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.
```

Before moving to the next step, remove any entries in the */etc/fstab* file for these filesystems so that the system does not attempt to mount them on the next reboot.

Once the disk is ready, the next step is to convert this disk into a physical volume using the *pvcreate* command (also wiping the dos signature if one exists):

```
# pvcreate /dev/sdb
WARNING: dos signature detected on /dev/sdb at offset 510. Wipe it? [y/n]: y
  Wiping dos signature on /dev/sdb.
  Physical volume "/dev/sdb" successfully created.
```

If the creation fails with a message that reads “Device /dev/<device> excluded by a filter”, it may be necessary to wipe the disk using the *wipefs* command before creating the physical volume:

```
# wipefs -a /dev/sdb
/dev/sdb: 8 bytes were erased at offset 0x00000200 (gpt): 45 46 49 20 50 41 52 54
/dev/sdb: 8 bytes were erased at offset 0x1fffffe00 (gpt): 45 46 49 20 50 41 52
54
/dev/sdb: 2 bytes were erased at offset 0x000001fe (PMBR): 55 aa
/dev/sdb: calling ioctl to re-read partition table: Success
```

With the physical volume created, we now need to add it to the volume group (in this case, *almalinux*) using the *vgextend* command:

```
# vgextend almalinux /dev/sdb
  Volume group "almalinux" successfully extended
```

The new physical volume has now been added to the volume group and is ready to be allocated to a logical volume. To do this, we run the *lvextend* tool providing the size by which we wish to extend the volume. In this case, we want to extend the size of the logical volume by 14 GB. Note that we need to provide the path to the logical volume, which can be obtained from the *lvdisplay* command (in this case, */dev/almalinux/home*):

```
# lvextend -L+14G /dev/almalinux/home
  Size of logical volume almalinux/home changed from <223.34 GiB (57174 extents)
to <237.34 GiB (60758 extents).
  Logical volume almalinux/home successfully resized.
```

The last step is to resize the file system residing on the logical volume to use the additional space. The way this is performed will depend on the filesystem type, which can be identified using the following *df* command and checking the Type column:

```
# df -T /home
Filesystem           Type 1K-blocks    Used Available Use% Mounted on
/dev/mapper/almalinux-home xfs  234070356 3345116 230725240   2% /home
```

If / is formatted using the XFS filesystem, it can be resized using the *xfs_growfs* utility:

```
# xfs_growfs /home
```

```
meta-data=/dev/mapper/almalinux-home isize=512    agcount=4, agsize=14636544 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=1        finobt=1, sparse=1, rmapbt=0
         =                       reflink=1    bigtime=1 inobtcount=1
data     =                       bsize=4096   blocks=58546176, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0, ftype=1
log      =internal log           bsize=4096   blocks=28587, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
data blocks changed from 58546176 to 62216192
```

If, on the other hand, the filesystem is of type ext2, ext3, or ext4, the *resize2fs* utility should be used instead when performing the filesystem resize:

```
# resize2fs /dev/almalinux/home
```

Once the resize completes, the file system will have been extended to use the additional space provided by the new disk drive. All this has been achieved without moving a single file or restarting the server. As far as users on the system are concerned, nothing has changed (except that there is now more disk space).

32.4 Summary

Volume groups and logical volumes provide an abstract layer on top of the physical storage devices on an AlmaLinux 9 system to provide a flexible way to allocate the space provided by multiple disk drives. This allows disk space allocations to be made and changed dynamically without the need to repartition disk drives and move data between filesystems. This chapter has outlined the basic concepts of volume groups and logical and physical volumes while demonstrating how to manage these using command-line tools.

Index

C

G

H

Q

R

S

X

Y