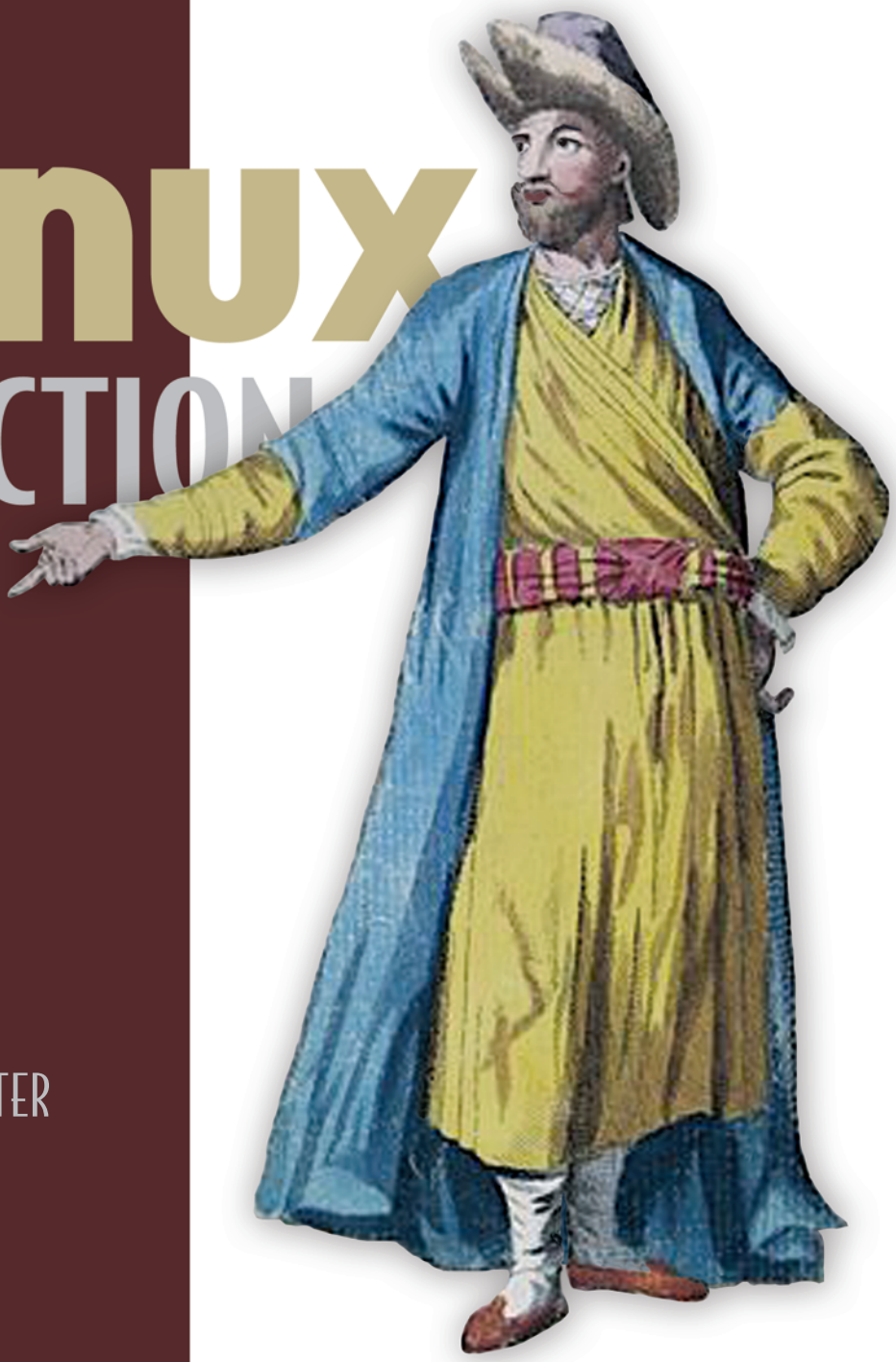
Linux IN ACTION



David Clinton

SAMPLE CHAPTER

MANNING

Linux in Action

by David Clinton

Chapter 2

brief contents

2 Linux virtualization: Building a Linux working environment

This chapter covers

- Finding the right virtualization technology
- Using Linux repository managers
- Building effective environments using VirtualBox
- Building containers with LXC
- How and when to closely manage VMs

Virtualization is the single most important technology behind almost all recent improvements in the way services and products are delivered. It's made entire industries from cloud computing to self-driving cars not only possible, but compelling. Curious? Here are two virtualization facts you'll need to know from the start:

- Linux absolutely dominates the virtual space.
- Virtualization makes it easier to learn any technology.

This chapter gives a good taste of the dominant enterprise virtualization technologies currently in use. But more to the point, it also enables you to use a virtualized environment where you can safely learn Linux administration skills. Why does this rather sophisticated technology show up so early in the book? Because it'll make it much easier for you to work through the rest of the chapters.

Need a fresh, clean operating system (OS) to try something new? Create one in a few seconds. Made a configuration error that's locked you out of your machine? No problem. Kill it and launch a new one. Along the way, you'll learn how to use Linux package managers to download, install, and manage all the software (like VirtualBox and LXC) that you'll need.

2.1 What is virtualization?

Once upon a time when you wanted a new server to provide some web server or document share for your company or its customers, you'd need to research, request budget approval, negotiate, order, safely house, provision, and then launch a brand-new machine. The process from start to finish could take months (trust me on that one—I've been there). And when increasing demand on that service threatened to overwhelm the server's capacity, you'd start the whole thing over again, hoping to eventually get the capacity/demand balance right.

A common scenario would see a company providing multiple but codependent services, each run on its own hardware. Picture a frontend web server deployed along with a database in the backend. When the dust settled, however, you'd often end up with one server deeply underused and one (usually right next to it on the rack) unable to keep up. But imagine you could securely share the compute, memory, storage, and networking resources of a single high-capacity server among multiple services. Imagine being able to carve virtual server instances out of that physical server by assigning them only the level of resources they need, and then instantly adjusting capacity to meet changing demands.

Now imagine being able to efficiently pack dozens of those virtual computers running multiple operating systems onto a single bare-metal server so that absolutely nothing is ever wasted. Imagine then being able to have those virtual machines (VMs) automatically spill over onto other physical servers as the first ones fill up. Imagine too the convenience of being able to kill a VM that's failed or in need of an update, and replace it so quickly that users might never realize anything has changed. Got that image in your head (hopefully, it's something like figure 2.1)? You're imagining *virtualization*.

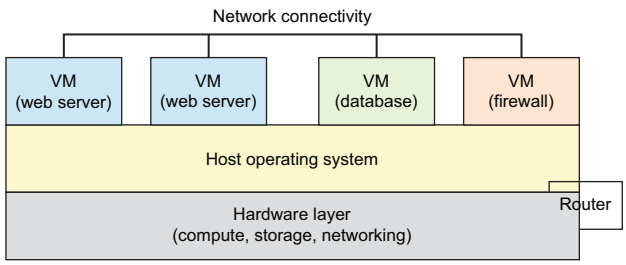


Figure 2.1 VM clients of a hardware host with connectivity to each other and to a larger network through an external router

That image is so attractive that it now dominates the enterprise computing world. At this point, I doubt there are many local or cloud-based server loads left that aren't running on some kind of virtualization technology. And the OS running the vast majority of those virtual workloads is Linux.

Amazon Web Services (AWS), by the way, lets customers rent capacity on (Linux) servers hosting millions of VMs that, in turn, run countless workloads, including many of the most popular online services. Figure 2.2 shows how an AWS Elastic Compute Cloud (EC2) VM instance serves as a hub for a full range of storage, database, and networking tools.

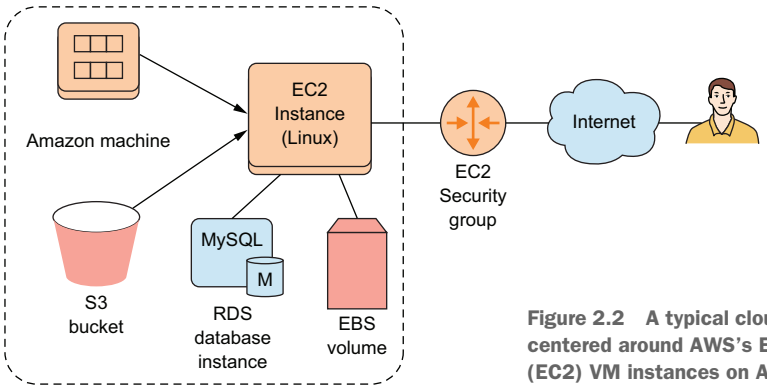


Figure 2.2 A typical cloud computing workload centered around AWS's Elastic Cloud Compute (EC2) VM instances on Amazon Web Services

Don't worry if some of those AWS details are a bit obscure—they're not the subject of this book in any case. But if you do find yourself wanting to learn more about Amazon Web Services, you could always read my book *Learn Amazon Web Services in a Month of Lunches* (Manning, 2017). And virtualization? There's my *Teach Yourself Linux Virtualization and High Availability* (LULU Press, 2017).

This next short section might feel a bit heavy, but it'll help provide some context for those of you interested in understanding how things are working under the hood. Successful virtualization uses some kind of isolated space on a physical computer where a guest OS can be installed and then fooled into thinking that it's all alone on its own computer. Guest operating systems can share network connections so that their administrators can log in remotely (something I'll discuss in chapter 3) and do their work exactly as they would on traditional machines. Those same shared network connections allow you to use the VMs to provide public services like websites. Broadly speaking, there are currently two approaches to virtualization:

- *Hypervisors*—Controls host system hardware to one degree or another, providing each guest OS the resources it needs (figure 2.3). Guest machines are run as system processes, but with virtualized access to hardware resources. AWS servers, for instance, have long been built on the open source Xen hypervisor technology (although they've recently begun switching some of their servers to the

equally open source KVM platform). Other important hypervisor platforms include VMware ESXi, KVM, and Microsoft's Hyper-V.

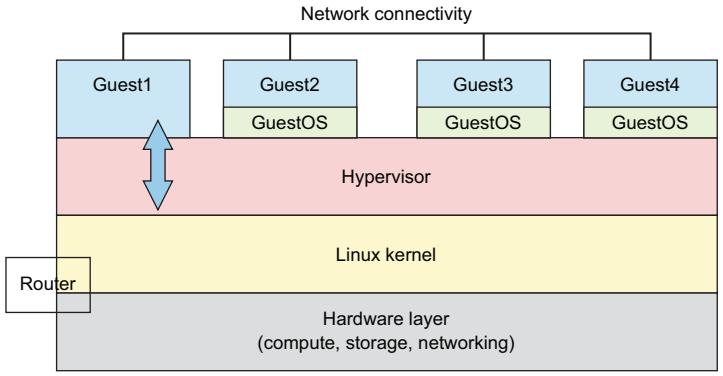


Figure 2.3 A type 2 hypervisor architecture showing full operating systems installed on each guest with some special administration duties delegated to Guest1

- *Containers*—Extremely lightweight virtual servers that, rather than running as full operating systems, share the underlying kernel of their host OS (see figure 2.4). Containers can be built from plain-text scripts, created and launched in seconds, and easily and reliably shared across networks. The best-known container technology right now is probably Docker. The Linux Container (LXC) project that we'll be working with in this chapter was Docker's original inspiration.

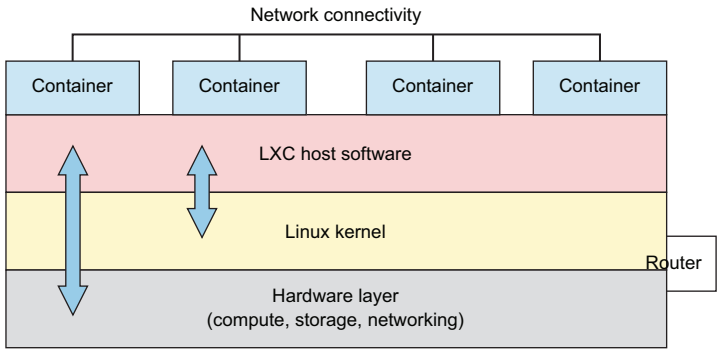


Figure 2.4 LXC architecture showing access between the LXC environment and both the Linux kernel and the hardware layer beneath it

No one technology is right for every project. But if you decide to hang around for the rest of this chapter, you're going to learn how and why to use two virtualization technologies: VirtualBox (a type 2 hypervisor) and, as I mentioned earlier, LXC (a container manager).

Design considerations

I wouldn't want you to walk away from this book without at least some basic guidelines for choosing virtualization technologies, so here are some thoughts:

- Full-sized hypervisors like Xen and KVM (through a management frontend like Libvirt) are normally used for enterprise-sized deployments involving large fleets of Linux VMs.
- VirtualBox (and VMware's Player) are perfect for testing and experimenting with live operating systems, one or two at a time, without the need to install them to actual PCs. Their relatively high overhead makes them unsuitable for most production environments.
- Container technologies like LXC and Docker are lightweight and can be provisioned and launched in mere seconds. LXC containers are particularly well suited to playing with new technologies and safely building OS software stacks. Docker is currently the technology running countless dynamic, integrated fleets of containers as part of vast microservices architectures. (I'll talk a bit more about microservices in chapter 9.)

2.2 Working with VirtualBox

There's a lot you can do with Oracle's open source VirtualBox. You can install it on any OS (including Windows) running on any desktop or laptop computer, or use it to host VM instances of almost any major OS.

Installing VirtualBox on a Windows PC

Want to try all this out from a Windows PC? Head over to the VirtualBox website (https://www.virtualbox.org/wiki/Downloads) and download the executable archive. Click the file you've downloaded, and then work through a few setup steps (the default values should all work). Finally, you'll be asked whether you're OK with a possible reset of your network interfaces and then whether you want to install VirtualBox. You do.

VirtualBox provides an environment within which you can launch as many virtual computers as your physical system resources can handle. And it's a particularly useful tool for safely testing and learning new administration skills, which is our primary goal right now. But before that'll happen, you need to know how downloading and installing software on Linux works.

2.2.1 Working with Linux package managers

Getting VirtualBox happily installed on an Ubuntu machine is simple. It takes two commands:

```
# apt update
# apt install virtualbox
```

NOTE Remember that the # prompt means this command requires admin privileges, which are normally accessed by prefacing the command with `sudo`.

But what happened in our example? It all revolves around the software package manager called Advanced Package Tool (APT, more commonly known as *apt*). In the Linux world, package managers connect computers to vast online repositories of thousands of software applications, most of them free and open source. The manager, which comes installed with Linux by default, has a number of jobs:

- Maintains a local index to track repositories and their contents
- Tracks the status of all the software that's installed on your local machine
- Ensures that all available updates are applied to installed software
- Ensures that software dependencies (other software packages or configuration parameters required by the package you're installing) are met for new applications before they're installed
- Handles installing and removing software packages

Figure 2.5 illustrates some elements of the ongoing relationship between an online software repository and the package manager running on a Linux computer.

The system works incredibly well and, for historical and economic reasons, there's nothing quite like it outside of the Linux world. The thing is, though, that the manager you use will depend on your particular Linux distribution. By and large, if your distribution falls within the Debian/Ubuntu family, then you'll use APT. Members of the Red Hat family will use the RPM manager and Yum (or its new DNF replacement). Table 2.1 shows a partial list of distributions.

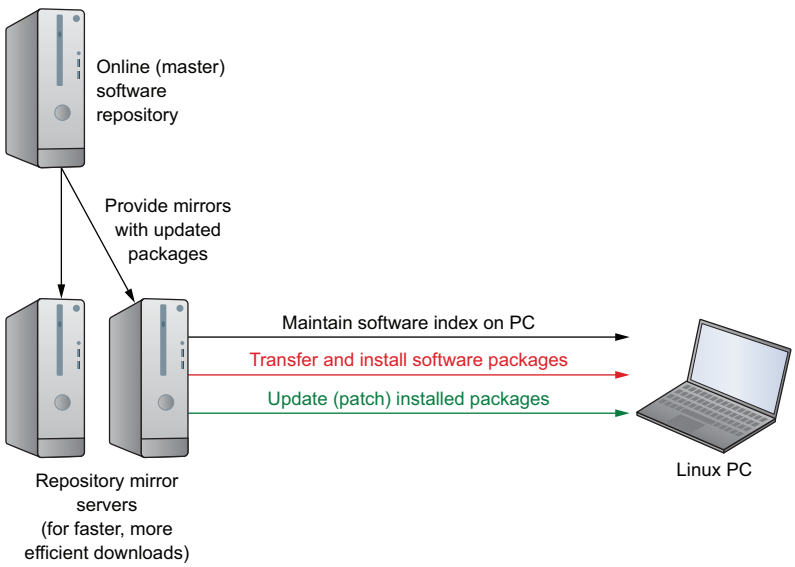


Figure 2.5 The relationships among master software repositories, mirror download servers, and Linux running on an end user machine

Table 2.1 Package managers and distros

Package manager	Distribution
APT	Debian
	Ubuntu
	Mint
	Kali Linux
RPM	Red Hat Enterprise Linux
	CentOS
	Fedora
YaST	SUSE Linux
	OpenSUSE

Besides using a package manager to install software from remote repositories, you may need to download software from a website. Often, you'll find packages that were formatted by their developers to work with APT or Yum once installed from the command line using a backend tool. Say, for instance, you want to use Skype. Heading over to its download page (figure 2.6) will let you download either a DEB or RPM file of the Skype for Linux package. You'd choose DEB if you're using a Debian-based distribution and APT, or RPM if you're in the Yum-loving Red Hat family.

WORKING WITH THE DEBIAN PACKAGE MANAGER

Once you download the file, you can install it from the command line using `dpkg`. Use the `-i` flag (for install). You'll need to make sure that you're running the `dpkg` command from the directory where the skypeforlinux-64 file is located. This example assumes that you saved the package to the Downloads directory in your user account:

```
$ cd /home/<username>/Downloads
# dpkg -i skypeforlinux-64.deb
```

The `dpkg` command should take care of dependencies for you. But if it doesn't, its output will usually give you enough information to know what's going on.

> **What's with that "-64"?**
>
> Linux, like other x86-based operating systems, comes in both 64-bit and 32-bit versions. The vast majority of computers manufactured and sold over the past decade use the faster 64-bit architecture. Because there's still older or development-oriented hardware out there, you'll sometimes need to run 32-bit, and you'll want the software you install to work with it.
>
> You can check for yourself by running `arch` from the command line. Unless you know you're running on older hardware (something Linux does particularly well, by the way), you're safe assuming that you're a 64-bit kind of person.

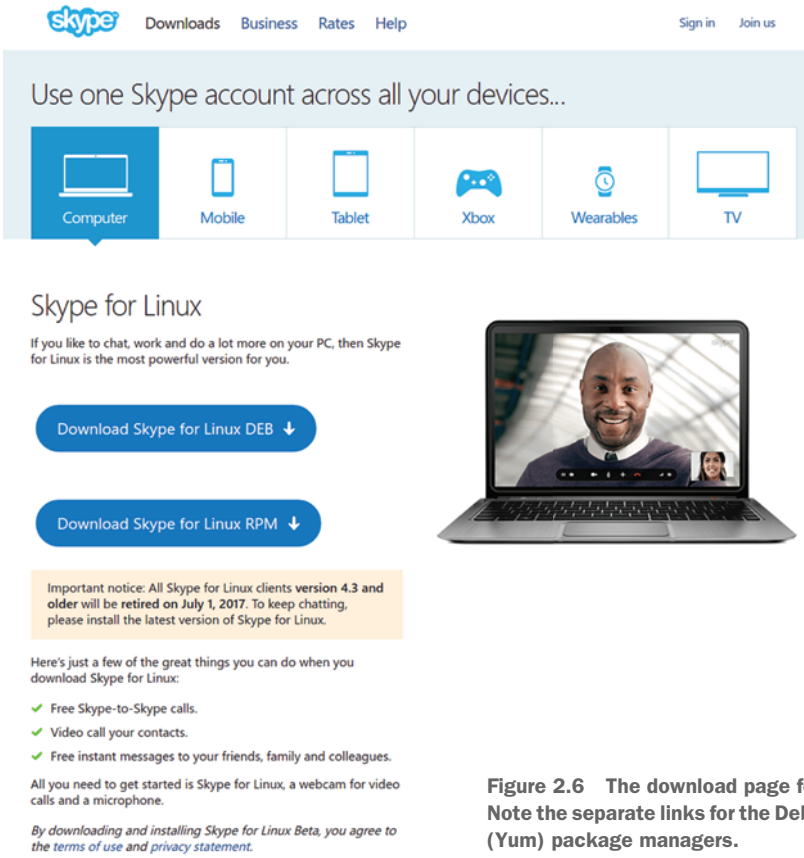


Figure 2.6 The download page for Skype for Linux. Note the separate links for the Debian (APT) and RPM (Yum) package managers.

Installing VirtualBox for the RPM package manager

Earlier, I introduced `apt update` and `apt install virtualbox`. What did those brief commands do? To explain, I'll install the same VirtualBox software on a machine running the Fedora Linux distribution. Because I'll use Red Hat's DNF package manager, it'll require a few extra steps—which is a good thing, because running them will illustrate how the process works. The process is a bit involved, so table 2.2 lists the steps.

Table 2.2 Steps to install VirtualBox on Fedora

Task	Command
Add repo	`wget http://download.virtualbox.org/virtualbox/rpm/fedora/virtualbox.repo`
Update index	`dnf update`
Install dependencies	`dnf install patch kernel-devel dkms`
Install package	`dnf install VirtualBox-5.1`

NOTE These steps were designed for and tested on Fedora version 25 and do a great job illustrating the package management process. It all might work more smoothly on more recent Fedora releases, though.

Back on Ubuntu, APT knew what I meant by *virtualbox* when I added it to the `install` command. That's because a VirtualBox package is part of an online repository with which APT is already familiar. It turns out, however, that Red Hat and its children (like CentOS and Fedora) aren't quite as sociable, at least not out of the box, so I'll need to add the virtualbox repository to Yum.

From the previous chapter, you'll remember that third-party software configuration files are often kept within the /etc/ directory hierarchy, and, in that respect, yum/DNF is no different. Repository information is kept in /etc/yum.repos.d/, so you should change to that directory. From there, you'll use the wget program (usually installed by default) to download the .repo file. Here's how to do all that:

```
$ cd /etc/yum.repos.d/
# wget http://download.virtualbox.org/virtualbox/rpm/fedora/
➥   virtualbox.repo
```

Installing software on Linux

Specific directions for installing Linux software, including details like the precise URL I used earlier, are almost always available online. You can find those either on the software developers' own websites or through freely available guides. The internet is your friend.

Make sure you specify the Linux distribution, release version, and architecture in your search engine phrases wherever necessary. I found details about the specific packages required for this project through my favorite search engine—so should you.

Having the .repo file in the right directory won't do much until you tell RPM what's changed. You do that by running `update`. The `update` command also checks the local repository index against its online counterparts to see whether there's anything new you'll want to know about. No matter what manager you're using, it's always a good idea to update the repo information before installing new software:

```
# dnf update
Importing GPG key 0x98AB5139:
 Userid     : "Oracle Corporation (VirtualBox archive signing key)
➥    <info@virtualbox.org>"
 Fingerprint: 7B0F AB3A 13B9 0743 5925 D9C9 5442 2A4B 98AB 5139
 From       : https://www.virtualbox.org/download/
➥     oracle_vbox.asc
Is this ok [y/N]: y
Fedora 25 - x86_64 - VirtualBox         120 kB/s |  33 kB     00:00
Dependencies resolved.
Nothing to do.
Complete!
```

All transactions with the repositories are encrypted using GPG keys.

The VirtualBox references refer to the fact that I'm running this Fedora host as a VM in VirtualBox.

The next step involves installing all the software dependencies that VirtualBox will need to run properly. A *dependency* is software that must already be installed on your computer for a new package to work. Back on Ubuntu, APT took care of these important details invisibly; Yum will also often take care of a lot of the backend details. But when it doesn't, forcing you to do it manually, the details are readily available from the same online sources discussed previously. Here's a truncated version of what that will look like:

```
# dnf install patch kernel-devel dkms
Last metadata expiration check: 0:43:23 ago on Tue Jun 13 12:56:16 2017.
[...]
Dependencies resolved.

================================================================================
 Package         Arch    Version                           Repository  Size
================================================================================
Installing:
dkms            noarch  2.3-5.20170523git8c3065c.fc25     updates     81 k
 kernel-devel    x86_64  4.11.3-202.fc25                   updates     11 M
 patch           x86_64  2.7.5-3.fc24                      fedora     125 k
Transaction Summary
================================================================================
Install  3 Packages
Total download size: 12 M
Installed size: 43 M
Is this ok [y/N]: y
Downloading Packages:
(1/3): dkms-2.3-5.20170523git8c3065c.fc25.noarc 382 kB/s |  81 kB     00:00
(2/3): patch-2.7.5-3.fc24.x86_64.rpm            341 kB/s | 125 kB     00:00
(3/3): kernel-devel-4.11.3-202.fc25.x86_64.rpm  2.4 MB/s |  11 MB     00:04
--------------------------------------------------------------------------------
Total                                           1.8 MB/s |  12 MB     00:06
[...]
Running transaction
  Installing  : kernel-devel-4.11.3-202.fc25.x86_64                       1/3
  Installing  : dkms-2.3-5.20170523git8c3065c.fc25.noarch                 2/3
  Installing  : patch-2.7.5-3.fc24.x86_64                                 3/3
  Verifying   : patch-2.7.5-3.fc24.x86_64                                 1/3
  Verifying   : kernel-devel-4.11.3-202.fc25.x86_64                       2/3
  Verifying   : dkms-2.3-5.20170523git8c3065c.fc25.noarch                 3/3
Installed:
  dkms.noarch 2.3-5.20170523git8c.fc25 kernel-devel.x86_64 4.11.3-202.fc25
  patch.x86_64 2.7.5-3.fc24
Complete!
```

Approve the operation by typing y before it will run.

A quick review of the successful operation

It's been a bit of a journey, but you're finally ready to install VirtualBox itself on your Red Hat, CentOS, or Fedora machine. The version number I used in this example came from the online guide used earlier. Naturally, by the time you get to try this out, it may no longer be 5.1.

DNF is obviously satisfied with the dependencies installed earlier.

```
# dnf install VirtualBox-5.1
Last metadata expiration check: 0:00:31 ago on Tue Jun 13 13:43:31 2017.
Dependencies resolved.
================================================================================
 Package            Arch    Version                    Repository     Size
================================================================================
Installing:
 SDL                x86_64  1.2.15-21.fc24             fedora        213 k
 VirtualBox-5.1     x86_64  5.1.22_115126_fedora25-1   virtualbox     68 M
 python-libs        x86_64  2.7.13-2.fc25              updates       6.2 M
 qt5-qtx11extras    x86_64  5.7.1-2.fc25               updates        30 k
Transaction Summary
================================================================================
Install  4 Packages
[...]
Is this ok [y/N]: y
[...]
Creating group 'vboxusers'. VM users must be member
➥      of that group!
[...]
Installed:
  SDL.x86_64 1.2.15-21.fc24
  VirtualBox-5.1.x86_64 5.1.22_115126_fedora25-1
  python-libs.x86_64 2.7.13-2.fc25
  qt5-qtx11extras.x86_64 5.7.1-2.fc25
Complete!
```

Returns a list of all the packages that will be installed in this operation

Note that the process created a new system group. I'll talk about groups in chapter 9.

VirtualBox add-ons

You should be aware that Oracle provides an Extension Pack for VirtualBox that adds features like USB support, disk encryption, and some alternatives to the existing boot options. Take those tools into account should you ever hit a dead end running the standard package.

You can also add extra file system and device integration between VirtualBox VMs and their host through the VBox Guest Additions CD-ROM image. This provides you with features like a shared clipboard and drag and drop. If the Vbox additions aren't already available through your host, install the Extension Pack on Ubuntu using this command:

```
sudo apt install virtualbox-guest-additions-iso
```

And then add it as a virtual optical drive to your running VM. Search online documentation concerning any extra packages that might be necessary for this to work on your host OS.

Before moving on to actually using virtualization tools like VirtualBox, I should leave you with at least a hint or two for tracking down other repository packages you might need. APT systems let you directly search for available packages using `apt search`.

This example searches for packages that might help you monitor your system health and then uses `apt show` to display full package information:

```
$ apt search sensors
$ apt show lm-sensors
```

The aptitude program, when installed, is a semi-graphic shell environment where you can explore and manage both available and already installed packages. If you can't live without your mouse, Synaptic is a full GUI package manager for desktop environments. And the Yum world is also fully searchable:

```
$ yum search sensors
$ yum info lm_sensors
```

2.2.2 Defining a virtual machine (VM)

I'm not sure whether you've ever put together a physical computer from components, but it can get involved. Defining a new VM within VirtualBox works pretty much the same way. The only significant difference is that, rather than having to get down on your hands and knees with a flashlight clenched between your teeth to manually add RAM and a storage drive to your box, VirtualBox lets you define your VM's "hardware" specs by clicking your mouse.

After clicking New in the VirtualBox interface, you'll give the VM you're about to build a descriptive name. As you can see in figure 2.7, the software should be able to correctly populate the Type and Version fields automatically. The Type and Version you select here won't install an actual OS, but are used to apply appropriate hardware emulation settings.

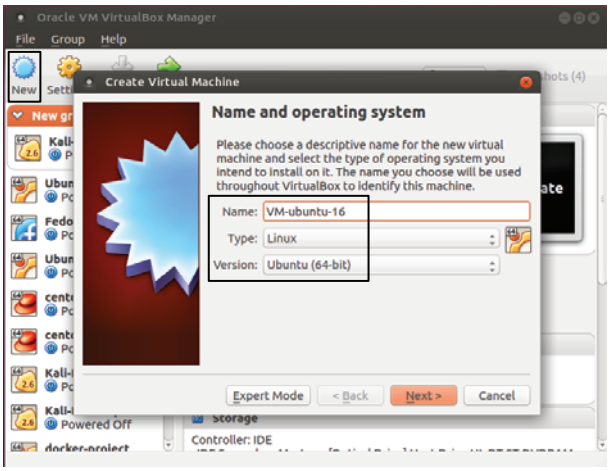


Figure 2.7 The Create Virtual Machine dialog: VirtualBox will try to approximate your OS and OS version to offer intelligent default choices later.

On the next screen, you'll allocate RAM to your VM. Unless you're planning something particularly demanding, like hosting a container swarm or running a busy web server, the default amount (768 MB) should be fine. You can certainly give it more RAM if necessary, but don't forget to leave enough for your host machine and any other VMs that might already live on it. If your host only has 4 GB of physical RAM, you probably won't want to give half of that to your VM.

Keep these limits in mind if you eventually decide to run multiple VMs at a time, something that will be useful for some of the projects you'll attempt later in the book. Even if each VM is only using the default amount of memory, two or three of them can start to eat away at the RAM needed for normal host operations.

The VirtualBox setup process now asks if you'd like to create a new virtual disk for your VM or use one that already exists (figure 2.8). What's a computer without a hard disk? There may be times when you want to share a single disk between two VMs, but for this exercise I'm guessing that you'll want to start from scratch. Select Create a Virtual Hard Disk Now.

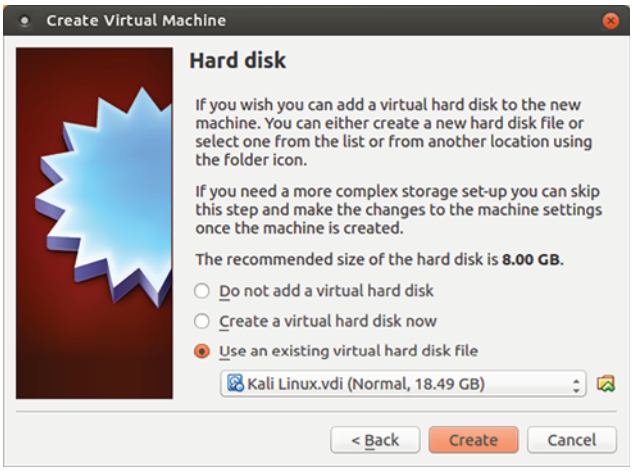


Figure 2.8 The Hard Disk screen. Note how, in this case, the non-default Use an Existing Virtual Hard Disk File radio button is selected.

The next screen (figure 2.9) lets you choose a hard disk file-type format for the disk you're about to create. Unless you're planning to eventually export the disk to use with some other virtualization environment, the default VirtualBox Disk Image (VDI) format will work fine.

I've also never regretted going with the default Dynamically Allocated option (figure 2.10) to determine how the virtual drive will consume space on the host. Here *dynamic* means space on the host storage disk will be allocated to the VM only as needed. Should the VM disk usage remain low, less host space will be allocated.

Figure 2.9 Virtual hard disks can be created using a number of formats. VDI is fine for VMs that will be used only within VirtualBox.

Figure 2.10 Dynamically allocated virtual disks will only consume as much space on their host's devices as they need.

A fixed-sized disk, on the other hand, will be given its maximum amount of space right away, regardless of how much it's actually using. The only advantage of Fixed Size is application performance. Because I generally only use VirtualBox VMs for testing and experiments, I'm fine avoiding the trade-off.

Because it knows it's Linux you're after, and because Linux makes such efficient use of storage space, VirtualBox will probably offer you only 8 GB of total disk size on the next screen (figure 2.11). Unless you've got unusually big plans for the VM (like, say, you're going to be working with some serious database operations), that will

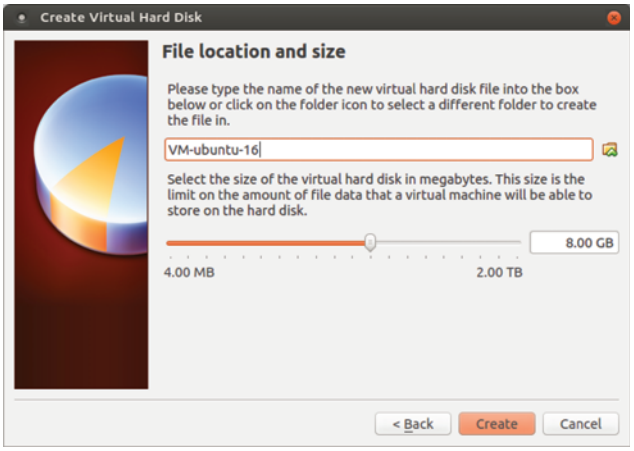


Figure 2.11 If necessary, your virtual disk can be as large as 2 TB or the maximum free space on the host device.

probably be fine. On the other hand, if you had chosen Windows as your OS, the default choice would have been 25 GB—and for good reason: Windows isn't shy about demanding lots of resources. That's a great illustration of one way Linux is so well suited to virtual environments.

> **NOTE** You can, if you like, also edit the name and location VirtualBox will use for your disk on the File Location and Size screen.

When you're done, click Create, and the new VM will appear in the list of VMs on the left side of the VirtualBox manager. But you're not done: that was the machine. Now you'll need an OS to bring it to life.

2.2.3 *Installing an operating system (OS)*

Now that you've defined your new VM's virtual hardware profile, here's what you'll still need to do:

1. Download a file (in ISO format) containing the image of the Linux distribution you want to use.
2. Boot the new VM using a virtual DVD drive containing the ISO you downloaded.
3. Work through the standard OS installation process.
4. Boot the VM and launch the OS you installed previously.

Once you've settled on a distribution, you'll need to download an .ISO file containing the OS files and installation program. Finding the right file is usually a matter of searching the internet for the distribution name and the word *download*. In the case of Ubuntu, you could alternatively go to the https://ubuntu.com page and click the Downloads tab as you see in figure 2.12. Notice the various flavors of Ubuntu that are

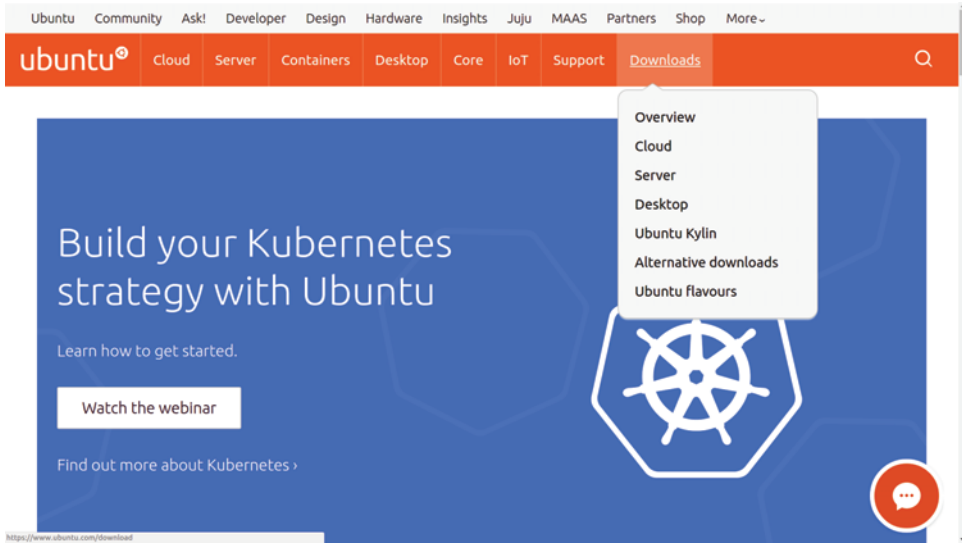


Figure 2.12 The Downloads drop-down on the home page of Ubuntu.com. Note the range of versions Ubuntu offers.

available. If you're going to be using this VM for administration tasks, then the small and fast Server version is probably a better choice than Desktop.

Large files can sometimes become corrupted during the download process. If even a single byte within your .ISO has been changed, there's a chance the installation won't work. Because you don't want to invest time and energy only to discover that there was a problem with the download, it's always a good idea to immediately calculate the checksum (or *hash*) for the .ISO you've downloaded to confirm that everything is as it was. To do that, you'll need to get the appropriate SHA or MD5 checksum, which is a long string looking something like this:

```
4375b73e3a1aa305a36320ffd7484682922262b3
```

You should be able to get this string from the same place you got your .ISO. In the case of Ubuntu, that would mean going to the web page at http://releases.ubuntu.com/, clicking the directory matching the version you've downloaded, and then clicking one of the links to a checksum (like, for instance, SHA1SUMS). You should compare the appropriate string from that page with the results of a command run from the same directory as your downloaded .ISO, which might look like this:

```
$ shasum ubuntu-16.04.2-server-amd64.iso
```

If they match, you're in business. If they don't (and you've double-checked to make sure you're looking at the right version), then you might have to download the .ISO a second time.

> **NOTE** You should be aware that there's more than one kind of hash. For many years, the MD5SUM algorithm was dominant, but SHA256 (with its longer 256-bit hashes) has been gaining in popularity. Practically, for large OS image files, one approach is probably no worse than the other.

Once your .ISO file is in place, head back to VirtualBox. With the VM you just created highlighted in the left panel, click the green Start button at the top of the app. You'll be prompted to select a .ISO file from your file system to use as a virtual DVD drive. Naturally, you'll choose the one you downloaded. The new VM will read this DVD and launch an OS installation.

Most of the time the installation process will go fine; however, including solutions to each of the many small things that could go wrong would require a couple of full chapters. If you do have trouble, you can consult the documentation and guides that are available for each Linux distro, or post your question on the *Linux in Action* forum on the Manning website and let the rest of us help.

When everything is nicely installed, there still might be a few more things to take care of before you can successfully boot into your VM. With your VM highlighted, click the yellow Settings icon. Here's where you can play with your VM's environment and hardware settings. Clicking Network, for example, allows you to define network connectivity. If you want your VM to have full internet access through the host machine's network interface, then, as shown in figure 2.13, you can select Bridged Adapter from the Attached To drop-down and then select the name of your host's adapter.

> **NOTE** Using a bridged adapter might not always be your first choice, and it might sometimes present a security risk. In fact, choosing NAT Network is a more common way to provide a VM with internet access. Because many of the exercises in this book require network connectivity between multiple VMs (something that's complicated using NAT), I'll go with a bridge for now.

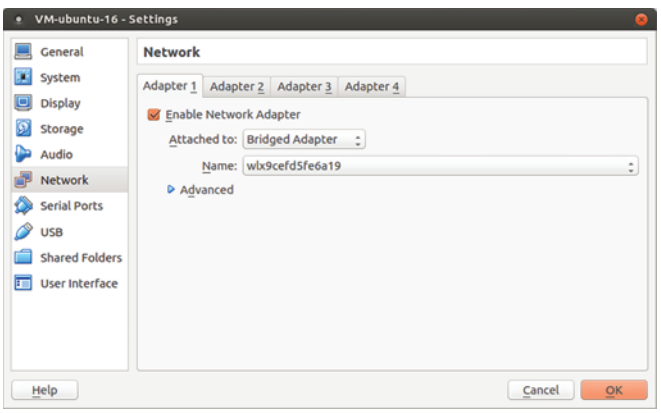


Figure 2.13 The Network tab of the Settings dialog where you can determine what type of network interface (or interfaces) to use for your VM

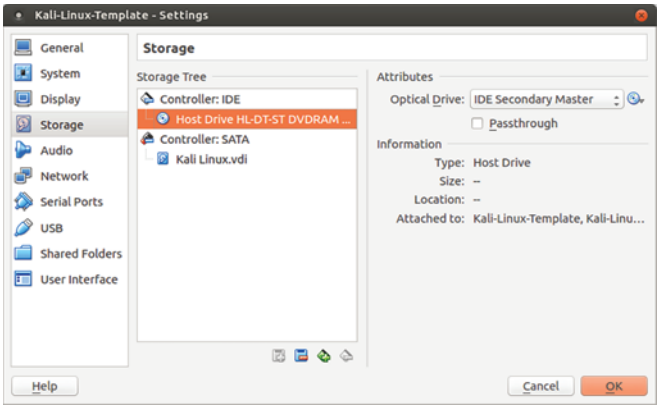


Figure 2.14 Remove a virtual disk by right-clicking its link and selecting Remove. You may need to do this to ensure that the VM boots to the right drive.

You may never need this next piece of information, but you'll appreciate knowing about it if you do. In some cases, to get the VM to boot properly, you'll also need to remove the DVD from the drive, like you would for a "real" physical installation. You do that by clicking Storage. Click the first disk listed and then the Remove Disk icon at the bottom (figure 2.14). Make sure you don't accidentally remove your hard disk!

It's also possible that you might sometimes need to mount a DVD (or .ISO file) to get VirtualBox to recognize it. Clicking the + icon with the Controller:IDE line highlighted lets you select a file to serve as a virtual optical drive.

2.2.4 Cloning and sharing a VirtualBox VM

This section is a bit bonus-y, but who doesn't like free stuff? I'm going to tell you about two related tricks: how to organize your VirtualBox VMs to make spinning up new ones as quick as possible and how to use the command line to share VMs across a network.

Cloning VMs for quick starts

One of the most obvious advantages of working with VMs is the ability to quickly access a fresh, clean OS environment. But if accessing that environment requires going through the full install process, than I don't see a whole lot of *quickly*, until you throw cloning into the mix. Why not keep your original VM in its clean post-install state and create an identical clone whenever you want to do some real work?

That's easy. Take another look at the VirtualBox app. Select the (*stopped*) VM you want to use as a master copy, click the Machine menu link, and then click Clone. You'll confirm the name you'd like to give your clone and then, after clicking Next, whether you want to create a Full Clone (meaning entirely new file copies will be created for the new VM) or Linked Clone (meaning the new VM will share all the base files with its master while maintaining your new work separately).

> **NOTE** Selecting the Linked option will go much faster and take up much less room on your hard disk. The only downside is that you'll be unable to move this particular clone to a different computer later. It's your choice.

Now click Clone, and a new VM appears in the VM panel. Start it the way you normally would, and then log in using the same credentials you set on the master.

MANAGING VMS FROM THE COMMAND LINE

VirtualBox comes with its own command-line shell that's invoked using `vboxmanage`. Why bother with the command line? Because, among other benefits, it allows you to work on remote servers, which can greatly increase the scope of possible projects. To see how `vboxmanage` works, use `list vms` to list all the VMs currently available on your system:

```
$ vboxmanage list vms
"Ubuntu-16.04-template" {c00d3b2b-6c77-4919-85e2-6f6f28c63d56}
"centos-7-template" {e2613f6d-1d0d-489c-8d9f-21a36b2ed6e7}
"Kali-Linux-template" {b7a3aea2-0cfb-4763-9ca9-096f587b2b20}
"website-project" {2387a5ab-a65e-4a1d-8e2c-25ee81bc7203}
"Ubuntu-16-lxd" {62bb89f8-7b45-4df6-a8ea-3d4265dfcc2f}
```

`vboxmanage clonevm` will pull off the same kind of clone action I described previously using the GUI. Here, I'm making a clone of the Kali Linux template VM, naming the copy newkali:

```
$ vboxmanage clonevm --register Kali-Linux-template --name newkali
```

You can verify that worked by running `vboxmanage list vms` once again.

That will work nicely as long as I only need to use the new VM here on my local computer. But suppose I wanted other members of my team to be able to run an exact copy of that VM, perhaps so they could test something I've been working on. For that, I'll need to convert the VM to some standardized file format. Here's how I might export a local VM to a file using the Open Virtualization Format:

```
$ vboxmanage export website-project -o website.ova
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
Successfully exported 1 machine(s).
```

The -o flag specifies the output filename: website.ova, in this case.

Next, you'll need to copy the .OVA file to your colleague's computer. Bear in mind that the file won't, by any standard, be considered small and dainty. If you haven't got network bandwidth to spare for a multiple-GB transfer, then consider moving it via a USB device. But if you do take the network route, the best tool for the job is Secure Copy (`scp`). Here's how that might work:

```
$ scp website.ova username@192.168.0.34:/home/username
```

If that whole `scp` thing seems a bit out-of-the-blue, don't worry: help is on the way. The `scp` command will be fully covered in chapter 3 as part of the OpenSSH content. In the meantime, that `scp` command will only work if OpenSSH is installed on both computers, you've authorized access to the username account on the remote computer, and it's reachable from your local machine.

Once the transfer is complete, all that's left is, from the remote computer, to import the VM into that machine's VirtualBox. The command is simple:

```
$ vboxmanage import website.ova
```

Confirm that the import operation worked using `list vms`, and try launching the VM from the desktop:

```
$ vboxmanage list vms
"website" {30ec7f7d-912b-40a9-8cc1-f9283f4edc61}
```

If you don't need fancy remote access, you can also share a VM from the GUI. With the machine you want to share highlighted, click the File menu in VirtualBox and then Export Appliance.

2.3 Working with Linux containers (LXC)

VirtualBox is great for running operations requiring Linux kernel access (the way you would if you were using security features like SELinux, as you'll see in chapter 9), for when you need GUI desktop sessions, or for testing operating systems like Windows. But if you need fast access to a clean Linux environment and you're not looking for any special release version, then you'd be hard pressed to beat LXC.

> **NOTE** Like any complex system, LXC might not work well with all hardware architectures. If you have trouble launching a container, consider the possibility that there might be a compatibility issue. The internet, as always should be a helpful source of deeper information.

How fast are LXC containers? You'll see for yourself soon enough. But because they skillfully share many system resources with both the host and other containers, they work like full-bore, standalone servers, using only minimal storage space and memory.

2.3.1 Getting started with LXC

Install LXC on your Ubuntu workstation? Piece of cake:

```
# apt update
# apt install lxc
```

Now how about on CentOS? Well, the cake is still there, but eating it will take a bit more work. That's partly because Ubuntu was built for and on Ubuntu and Debian. By all means, give it a shot on CentOS, but I won't guarantee success. You'll first need to

add a new repository, Extra Packages for Enterprise Linux (EPEL), and then install LXC along with some dependencies:

```
# yum install epel-release
# yum install lxc lxc-templates libcap-devel \
 libcgroup busybox wget bridge-utils lxc-extra libvirt
```

The backslash character (\) can be used to conveniently break a long command into multiple lines on the command line.

That's it. You're ready to get down to business. The basic LXC skill set is actually quite simple. I'm going to show you the three or four commands you'll need to make it all work, and then an insider tip that, once you understand how LXC organizes itself, will blow you away.

2.3.2 Creating your first container

Why not dive right in and create your first container? The value given to `-n` sets the name you'll use for the container, and `-t` tells LXC to build the container with the Ubuntu template:

```
# lxc-create -n myContainer -t ubuntu
```

The create process can take a few minutes to complete, but you'll see verbose output and, eventually, a success notification displayed to the terminal.

> **NOTE** You'll probably start seeing references to an alternate set of `lxc` commands associated with the relatively new LXD container manager. LXD still uses LXC tools under the hood but through a slightly different interface. As an example, using LXD the previous command would look like this: `lxc launch ubuntu:16.04 myContainer`. Both command sets will continue to be widely available.

There are actually quite a few templates available, as you can see from this listing of the /usr/share/lxc/templates/ directory:

```
$ ls /usr/share/lxc/templates/
lxc-alpine      lxc-centos     lxc-fedora         lxc-oracle       lxc-sshd
lxc-altlinux    lxc-cirros     lxc-gentoo         lxc-plamo        lxc-ubuntu
lxc-archlinux   lxc-debian     lxc-openmandriva   lxc-slackware    lxc-ubuntu-cloud
lxc-busybox     lxc-download   lxc-opensuse       lxc-sparclinux
```

> **WARNING** Not all of these templates are guaranteed to work right out of the box. Some are provided as experiments or *works in progress*. Sticking with the Ubuntu template on an Ubuntu host is probably a safe choice. As I noted, historically, LXC has always worked best on Ubuntu hosts. Your mileage may vary when it comes to other distros.

If you decided to create, say, a CentOS container, then you should make a note of the final few lines of the output, as it contains information about the password you'd use to log in:

```
# lxc-create -n centos_lxc -t centos
[...]
The temporary root password is stored in:
        '/var/lib/lxc/centos_lxc/tmp_root_pass'
```

In this example, I called the container centos_lxc and used the centos template.

The root password is located in a directory named after the container.

You'll log in using the user name *root* and the password contained in the tmp_root_pass file. If, on the other hand, your container uses the Ubuntu template, then you'll use *ubuntu* for both your user name and password. Naturally, if you plan to use this container for anything serious, you'll want to change that password right away:

```
$ passwd
Changing password for ubuntu.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

By the way, that command is, in fact, `passwd` and not `password`. My guess is that the creator of the passwd program didn't like typing. Now use `lxc-ls --fancy` to check the status of your container:

```
# lxc-ls --fancy
NAME    STATE   AUTOSTART GROUPS IPV4    IPV6
myContainer  STOPPED 0         -      -       -
```

Well, it exists, but apparently it needs starting. As before, the `-n` specifies by name the container you want to start. The `-d` stands for detach, meaning you don't want to be automatically dropped into an interactive session as the container starts. There's nothing wrong with interactive sessions: some of my best friends are interactive sessions, in fact. But in this case, running the `lxc-start` command without `-d` would mean that the only way to get out would involve shutting down the container, which might not be what you're after:

```
# lxc-start -d -n myContainer
```

Listing your containers should now display something like this:

```
# lxc-ls --fancy
NAME    STATE   AUTOSTART GROUPS IPV4        IPV6
myContainer  RUNNING 0       -    10.0.3.142   -
```

The container state is now RUNNING.

This time, the container is running and has been given an IP address (10.0.3.142). You could use this address to log in using a secure shell session, but not before reading

chapter 3. For now, you can launch a root shell session within a running container using `lxc-attach`:

```
# lxc-attach -n myContainer
root@myContainer:/#
```

Note the information in the new command prompt.

You might want to spend a couple of minutes checking out the neighborhood. For instance, `ip addr` will list the container's network interfaces. In this case, the `eth0` interface has been given an IP address of 10.0.3.142, which matches the IPV4 value received from `lxc-ls --fancy` earlier:

```
root@myContainer:/# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue
➥         state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
10: eth0@if11: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
➥       state UP group default qlen 1000
    link/ether 00:16:3e:ab:11:a5 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.3.142/24 brd 10.0.3.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::216:3eff:feab:11a5/64 scope link
       valid_lft forever preferred_lft forever
```

eth0 is, in this case, the designation for the container's primary network interface.

The container's IP address (10.0.3.142) and CIDR netmask (/24)

When you're done looking around your new container, you can either run `exit` to log out leaving the container running

```
root@myContainer:/# exit
exit
```

or shut down the container using `shutdown -h now`. But before you do that, let's find out how blazing fast LXC containers are. The `-h` flag I added to `shutdown` before stands for *halt*. If I used `r` instead, rather than shutting down for good, the container would reboot. Let's run `reboot` and then try to log in again right away to see how long it takes for the container to get back up on its feet:

```
root@myContainer:/# shutdown -r now
# lxc-attach -n myContainer
```

How did that go? I'll bet that by the time you managed to retype the `lxc-attach` command, myContainer was awake and ready for action. Did you know that pressing the up arrow key in Bash will populate the command line with the previous command? Using that would make it even faster to request a log in. In my case, there was no noticeable delay. The container shut down and fully rebooted in less than 2 seconds!

> **NOTE** LXC containers are also easy on system resources. Unlike my experience with VirtualBox VMs, where running three concurrently already starts to seriously impact my 8 GB host workstation performance, I can launch all kinds of LXC containers without suffering a slowdown.

What's that you say? How about that insider tip I promised you? Excellent. I can see you're paying attention. Well, back in a terminal on the host machine (as opposed to the container), you'll need to open an administrator shell using `sudo su`. From here on, until you type `exit`, you'll be *sudo* full time:

```
$ sudo su
[sudo] password for username:
#
```

Now change directory to /var/lib/lxc/, and list the contents. You should see a directory with the name of your container. If you've got other containers on the system, they'll have their own directories as well:

```
# cd /var/lib/lxc
# ls
myContainer
```

Move to your container directory, and list its contents. There'll be a file called config and a directory called rootfs (the *fs* stands for file system):

```
# cd myContainer
# ls
config rootfs
```

Feel free to take a look through config: that's where the basic environment values for the container are set. Once you're a bit more comfortable with the way LXC works, you'll probably want to use this file to tweak the way your containers behave. But it's the rootfs directory that I really wanted to show you:

```
# cd rootfs
# ls
bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr
```

All those subdirectories that fill rootfs, do they look familiar to you? They're all part of the Linux Filesystem Hierarchy Standard (FHS). This is the container's root (/) directory but within the host's file system. As long as you have admin permissions on the host, you'll be able to browse through those directories and edit any files you want—even when the container isn't running.

You'll be able to do all kinds of things with this access, but here's one that can quite possibly save your (professional) life one day. Suppose you lock yourself out on a container. Now there's nothing stopping you from navigating through the file system,

fixing the configuration file that you messed up, and getting back to work. Go ahead, tell me that's not cool. But it gets better.

It's true that the Docker ecosystem has gained many layers of features and sophistication since the technology moved out from under LXC's shadow some years ago. Under the hood, however, it's still built on top of a basic structural paradigm that will be instantly recognizable to anyone familiar with LXC. This means, should you be inclined to test the waters with the fastest-growing virtualization technology of the decade, you've already got skin in the game.

Summary

- Hypervisors like VirtualBox provide an environment where virtual operating systems can safely access hardware resources, whereas lightweight containers share their host's software kernel.
- Linux package managers like APT and RPM (Yum) oversee the installation and administration of software from curated online repositories using a regularly updated index that mirrors the state of the remote repository.
- Getting a VM going in VirtualBox requires defining its virtual hardware environment, downloading an OS image, and installing the OS on your VM.
- You can easily clone, share, and administer VirtualBox VMs from the command line.
- LXC containers are built on predefined, distribution-based templates.
- LXC data is stored within the host file system, making it easy to administer containers.

Key terms

- *Virtualization* is the logical sharing of compute, storage, and networking resources among multiple processes, allowing each to run as if it was a standalone physical computer.
- A *hypervisor* is software running on a host machine that exposes system resources to a guest layer, allowing the launching and administration of full-stack guest VMs.
- A *container* is a VM that, instead of full-stack, lives on top of (and shares) the host machine's core OS kernel. Containers are extremely easy to launch and kill, according to short-term need.
- A *dynamically allocated* virtual drive in VirtualBox takes up only as much space on your physical drives as the VM actually uses. A *fixed-size* disk, by contrast, takes up the maximum space no matter how much data is there.
- A *software repository* is a location where digital resources can be stored. Repositories are particularly useful for collaboration and distribution of software packages.

Security best practices

- Allowing an official package manager to install and maintain the software on your Linux system is preferred over doing it manually. Online repositories are much more secure, and downloading is properly encrypted.
- Always scan the checksum hashes of downloaded files against the correct hash strings, not only because packages can be corrupted during download, but because they can also sometimes be switched by man-in-the-middle attackers.

Command-line review

- `apt install virtualbox` uses APT to install a software package from a remote repository.
- `dpkg -i skypeforlinux-64.deb` directly installs a downloaded Debian package on a Ubuntu machine.
- `wget https://example.com/document-to-download` uses the wget command-line program to download a file.
- `dnf update`, `yum update`, or `apt update` syncs the local software index with what's available from online repositories.
- `shasum ubuntu-16.04.2-server-amd64.iso` calculates the checksum for a downloaded file to confirm that it matches the provided value. This means that the contents haven't been corrupted in transit.
- `vboxmanage clonevm Kali-Linux-template --name newkali` uses the vboxmanage tool to clone an existing VM.
- `lxc-start -d -n myContainer` starts an existing LXC container.
- `ip addr` displays information on each of a system's network interfaces (including their IP addresses).
- `exit` leaves a shell session without shutting down the machine.

Test yourself

1. Which of the following is a quality shared by both containers and hypervisors?
   a. They both allow VMs to run independently of the host OS.
   b. They both rely on the host's kernel for their basic operations.
   c. They both permit extremely lightweight VMs.
   d. They both permit extremely efficient use of hardware resources.
2. Which of the following is *not* the responsibility of a Linux package manager?
   a. Sync the local index with remote repositories.
   b. Scan installed software for malware.
   c. Apply updates to installed software.
   d. Ensure all package dependencies are installed.

3 Which of the following commands would you use to directly install a downloaded software package on a Ubuntu system?
   a `dpkg -i`
   b `dnf --install`
   c `apt install`
   d `yum -i`
4 When creating a VM on VirtualBox, which of the following steps comes first?
   a Select a hard disk file type.
   b Choose between Dynamically Allocated and Fixed Size.
   c Remove the virtual DVD from the drive.
   d Configure the network interface.
5 Which of the following formats can be used for OS images?
   a VDI
   b VMI
   c ISO
   d VMDK
6 Which of the following commands would you use to save a VM to a .OVA formatted file?
   a `vboxmanage export`
   b `vboxmanage clonevm`
   c `vboxmanage import`
   d `vboxmanage clone-ova`
7 Which of the following LXC command-line flags will start a container without automatically opening a new shell session?
   a `lxc-start -t`
   b `lxc-start -a`
   c `lxc-start -d`
   d `lxc-start -n`
8 By default, in which of the following directories will you find a container's file system?
   a /usr/share/lxc/
   b /etc/share/lxc/
   c /usr/lib/lxc/
   d /var/lib/lxc/

Answer key

1. d, 2. b, 3. a, 4. a, 5. c, 6. a, 7. c, 8. d

LINUX

Linux IN ACTION

David Clinton

You can't learn anything without getting your hands dirty—including Linux. Skills like securing files, folders, and servers, safely installing patches and applications, and managing a network are required for any serious user, including developers, administrators, and DevOps professionals. With this hands-on tutorial, you'll roll up your sleeves and learn Linux project by project.

Linux in Action guides you through 12 real-world projects, including automating a backup-and-restore system, setting up a private Dropbox-style file cloud, and building your own MediaWiki server. You'll try out interesting examples as you lock in core practices like virtualization, disaster recovery, security, backup, DevOps, and system troubleshooting. Each chapter ends with a review of best practices, new terms, and exercises.

What's Inside

- Setting up a safe Linux environment
- Managing secure remote connectivity
- Building a system recovery device
- Patching and upgrading your system

No prior Linux admin experience is required.

David Clinton is a certified Linux Server Professional, seasoned instructor, and author of Manning's bestselling *Learn Amazon Web Services in a Month of Lunches*.

To download their free eBook in PDF, ePub, and Kindle formats, owners of this book should visit manning.com/books/linux-in-action

"An essential guide to understanding Linux—with plenty of real-world examples."
—Dario Victor Durán
HiQ Stockholm

"Teaches a wide variety of Linux features that will make your life that much easier. Practical."
—Jens Christian B. Madsen
IT Relation

"The go-to book for Linux system administration."
—Gustavo Patino
Oakland University William Beaumont School of Medicine

"Everything you need to start maintaining Linux. It's not about how to use Linux, but how to take care of it."
—Maciej Jurkowski, Grupa Pracuj

ISBN-13: 978-1-61729-493-8
ISBN-10: 1-61729-493-4
53999
9 781617 294938

MANNING

$39.99 / Can $52.99 [INCLUDING eBOOK]