Chapter 1: Understanding Linux Virtualization

Virtualization is the technology that started a big technology shift toward IT consolidation, which provides more efficient use of resources and the cloud as a more integrated, automated, and orchestrated version of virtualization with a focus on not only virtual machines but also additional services. There are a total of 16 chapters in this book, all of which have been lined up to cover all the important aspects of Kernel-based Virtual Machine (KVM) virtualization. We will start with basic KVM topics such as the history of virtualization concepts and Linux virtualization and then move on and look at advanced topics in KVM such as automation, orchestration, virtual networking, storage, and troubleshooting. This chapter will provide you with an insight into the prevailing technologies in Linux virtualization and their advantages over others.

In this chapter, we will cover the following topics:

- Linux virtualization and its basic concepts
- Types of virtualization
- Hypervisor/VMM
- Open source virtualization projects
- What Linux virtualization offers you in the cloud

Linux virtualization and how it all started

Virtualization is a concept that creates virtualized resources and maps them to physical resources. This process can be done using specific hardware functionality (partitioning, via some kind of partition controller) or software functionality (hypervisor). So, as an example, if you have a physical PC-based server with 16 cores running a hypervisor, you can easily create one or more virtual machines with two cores each and start them up. Limits regarding how many

virtual machines you can start is something that's vendor-based. For example, if you're running Red Hat Enterprise Virtualization v4.x (a KVM-based bare-metal hypervisor), you can use up to 768 logical CPU cores or threads (you can read more information about this at **https://access.redhat.com/articles/906543**). In any case, hypervisor is going to be the *go-to guy* that's going to try to manage that as efficiently as possible so that all of the virtual machine workloads get as much time on the CPU as possible.

I vividly remember writing my first article about virtualization in 2004. AMD just came out with its first consumer 64-bit CPUs in 2003 (Athlon 64, Opteron) and it just threw me for a loop a bit. Intel was still a bit hesitant to introduce a 64-bit CPU – a lack of a 64-bit Microsoft Windows OS might have had something to do with that as well. Linux was already out with 64-bit support, but it was a dawn of many new things to come to the PC-based market. Virtualization as such wasn't something revolutionary as an idea since other companies already had non-x86 products

that could do virtualization for decades (for example, IBM CP-40 and its S/360-40, from 1967). But it sure was a new idea for a PC market, which was in a weird phase with many things happening at the same time. Switching to 64-bit CPUs with multi-core CPUs appearing on the market, then switching from DDR1 to DDR2, and then from PCI/ISA/AGP to PCI Express, as you might imagine, was a challenging time.

Specifically, I remember thinking about the possibilities – how cool it would be to run an OS, and then another couple of OSes on top of that. Working in the publishing industry, you might imagine how many advantages that would offer to anyone's workflow, and I remember really getting excited about it.

15 or so years of development later, we now have a competitive market in terms of virtualization solutions – Red Hat with KVM, Microsoft with Hyper-V, VMware with ESXi, Oracle with Oracle VM, and Google and other key players duking it out for users and market dominance.

This led to the development of various cloud solutions such as EC2, AWS, Office 365, Azure, vCloud Director, and vRealize Automation for various types of cloud services. All in all, it was a very productive 15 years for IT, wouldn't you say?

But, going back to October 2003, with all of the changes that were happening in the IT industry, there was one that was really important for this book and virtualization for Linux in general: the introduction of the first open source Hypervisor for x86 architecture, called **Xen**. It supports various CPU architectures (Itanium, x86, x86_64, and ARM), and it can run various OSes – Windows, Linux, Solaris, and some flavors of BSD – and it's still alive and kicking as a virtualization solution of choice for some vendors, such as Citrix (XenServer) and Oracle (Oracle VM). We'll get into more technical details about Xen a little bit later in this chapter.

The biggest corporate player in the open source market, Red Hat, included Xen virtualization in

initial releases of its Red Hat Enterprise Linux 5, which was released in 2007. But Xen and Red Hat weren't exactly a match made in heaven and although Red Hat shipped Xen with its Red Hat Enterprise Linux 5 distribution, Red Hat switched to **KVM** in Red Hat Enterprise Linux 6 in 2010, which was – at the time – a very risky move. Actually, the whole process of migrating from Xen to KVM began in the previous version, with 5.3/5.4 releases, both of which came out in 2009. To put things into context, KVM was a pretty young project back then, just a couple of years old. But there were more than a few valid reasons why that happened, varying from *Xen is not in the mainline kernel, KVM is*, to political reasons (Red Hat wanted more influence over Xen development, and that influence was fading with time).

Technically speaking, KVM uses a different, modular approach that transforms Linux kernels into fully functional hypervisors for supported CPU architectures. When we say *supported CPU architectures*, we're talking about the basic require-

ment for KVM virtualization – CPUs need to support hardware virtualization extensions, known as AMD-V or Intel VT. To make things a bit easier, let's just say that you're really going to have to try very hard to find a modern CPU that doesn't support these extensions. For example, if you're using an Intel CPU on your server or desktop PC, the first CPUs that supported hardware virtualization extensions date all the way back to 2006 (Xeon LV) and 2008 (Core i7 920). Again, we'll get into more technical details about KVM and provide a comparison between KVM and Xen a little bit later in this chapter and in the next.

Types of virtualization

There are various types of virtualization solutions, all of which are aimed at different use cases and are dependent on the fact that we're virtualizing a different piece of the hardware or software stack, that is, *what* you're virtualizing. It's also worth noting that there are different types of virtualization in terms of *how* you're vir-

tualizing – by partitioning, full virtualization, paravirtualization, hybrid virtualization, or container-based virtualization.

So, let's first cover the five different types of virtualization in today's IT based on *what* you're virtualizing:

- **Desktop virtualization** (**Virtual Desktop Infrastructure** **(VDI)**): This is used by a lot of enterprise companies and offers huge advantages for a lot of scenarios because of the fact that users aren't dependent on a specific device that they're using to access their desktop system. They can connect from a mobile phone, tablet, or a computer, and they can usually connect to their virtualized desktop from anywhere as if they're sitting at their workplace and using a hardware computer. Benefits include easier, centralized management and monitoring, much more simplified update workflows (you can update the base image for hundreds of virtual machines in a VDI solution and re-link that to hundreds of

virtual machines during maintenance hours), simplified deployment processes (no more physical installations on desktops, workstations, or laptops, as well as the possibility of centralized application management), and easier management of compliance and security-related options.

- **Server virtualization**: This is used by a vast majority of IT companies today. It offers good consolidation of server virtual machines versus physical servers, while offering many other operational advantages over regular, physical servers – easier to backup, more energy efficient, more freedom in terms of moving workloads from server to server, and more.
- **Application virtualization**: This is usually implemented using some kind of streaming/remote protocol technology such as Microsoft App-V, or some solution that can package applications into volumes that can be mounted to the virtual machine and profiled

for consistent settings and delivery options, such as VMware App Volumes.

- **Network virtualization** (and a more broader, cloud-based concept called **Software-Defined Networking (SDN)**): This is a technology that creates virtual networks that are independent of the physical networking devices, such as switches. On a much bigger scale, SDN is an extension of the network virtualization idea that can span across multiple sites, locations, or data centers. In terms of the concept of SDN, entire network configuration is done in software, without you necessarily needing a specific physical networking configuration. The biggest advantage of network virtualization is how easy it is for you to manage complex networks that span multiple locations without having to do massive, physical network reconfiguration for all the physical devices on the network data path. This concept will be explained in ***Chapter 4***, *libvirt Networking*, and ***Chapter 12***, *Scaling Out KVM with OpenStack*.

- **Storage virtualization** (and a newer concept **Software-Defined Storage (SDS)**): This is a technology that creates virtual storage devices out of pooled, physical storage devices that we can centrally manage as a single storage device. This means that we're creating some sort of abstraction layer that's going to isolate the internal functionality of storage devices from computers, applications, and other types of resources. SDS, as an extension of that, *decouples* the storage software stack from the hardware it's running on by abstracting control and management planes from the underlying hardware, as well as offering different types of storage resources to virtual machines and applications (block, file, and object-based resources).

If you take a look at these virtualization solutions and scale them up massively (hint: the cloud), that's when you realize that you're going to need various tools and solutions to *effectively* manage the ever-growing infrastructure, hence the development of various automatization and orchestration tools. Some of these tools will be covered

later in this book, such as Ansible in ***Chapter 11***, *Ansible for Orchestration and Automation*. For the time being, let's just say that you just can't manage an environment that contains thousands of virtual machines by relying on standard utilities only (scripts, commands, and even GUI tools). You're definitely going to need a more programmatic, API-driven approach that's tightly integrated with the virtualization solution, hence the development of OpenStack, OpenShift, Ansible, and the **Elasticsearch, Logstash, Kibana** (**ELK**) stack, which we'll cover in ***Chapter 14***, *Monitoring the KVM Virtualization Platform Using the ELK Stack*.

If we're talking about *how* we're virtualizing a virtual machine as an object, there are different types of virtualization:

- **Partitioning**: This is a type of virtualization in which a CPU is divided into different parts, and each part works as an individual system. This type of virtualization solution isolates a server into partitions, each of which can run a

separate OS (for example, **IBM Logical Partitions (LPARs)**).

- **Full virtualization**: In full virtualization, a virtual machine is used to simulate regular hardware while not being aware of the fact that it's virtualized. This is done for compatibility reasons – we don't have to modify the guest OS that we're going to run in a virtual machine. We can use a software- and hardware-based approach for this.
  Software-based: Uses binary translation to virtualize the execution of sensitive instruction sets while emulating hardware using software, which increases overhead and impacts scalability.
  Hardware-based: Removes binary translation from the equation while interfacing with a CPU's virtualization features (AMD-V, Intel VT), which, in turn, means that instruction sets are being executed directly on the host CPU. This is what KVM does (as well as other popular hypervisors, such as ESXi, Hyper-V, and Xen).

- **Paravirtualization**: This is a type of virtualization in which the guest OS understands the fact that it's being virtualized and needs to be modified, along with its drivers, so that it can run on top of the virtualization solution. At the same time, it doesn't need CPU virtualization extensions to be able to run a virtual machine. For example, Xen can work as a paravirtualized solution.
- **Hybrid virtualization**: This is a type of virtualization that uses full virtualization and paravirtualization's biggest virtues – the fact that the guest OS can be run unmodified (full), and the fact that we can insert additional paravirtualized drivers into the virtual machine to work with some specific aspects of virtual machine work (most often, I/O-intensive memory workloads). Xen and ESXi can also work in hybrid virtualization mode.
- **Container-based virtualization**: This is a type of application virtualization that uses containers. A container is an object that packages an application and all its dependencies so that the

application can be scaled out and rapidly deployed without needing a virtual machine or a hypervisor. Keep in mind that there are technologies that can operate as both a hypervisor and a container host at the same time. Some examples of this type of technology include Docker and Podman (a replacement for Docker in Red Hat Enterprise Linux 8).

Next, we're going to learn how to use hypervisors.

Using the hypervisor/virtual machine manager

As its name suggests, the **Virtual Machine Manager (VMM)** or hypervisor is a piece of software that is responsible for monitoring and controlling virtual machines or guest OSes. The hypervisor/VMM is responsible for ensuring different virtualization management tasks, such as providing virtual hardware, virtual machine life cycle management, migrating virtual machines,

allocating resources in real time, defining policies for virtual machine management, and so on. The VMM/hypervisor is also responsible for efficiently controlling physical platform resources, such as memory translation and I/O mapping. One of the main advantages of virtualization software is its capability to run multiple guests operating on the same physical system or hardware. These multiple guest systems can be on the same OS or different ones. For example, there can be multiple Linux guest systems running as guests on the same physical system. The VMM is responsible for allocating the resources requested by these guest OSes. The system hardware, such as the processor, memory, and so on, must be allocated to these guest OSes according to their configuration, and the VMM can take care of this task. Due to this, the VMM is a critical component in a virtualization environment.

In terms of types, we can categorize hypervisors as either type 1 or type 2.

Type 1 and type 2 hypervisors

Hypervisors are mainly categorized as either type 1 or type 2 hypervisors, based on where they reside in the system or, in other terms, whether the underlying OS is present in the system or not. But there is no clear or standard definition of type 1 and type 2 hypervisors. If the VMM/hypervisor runs directly on top of the hardware, its generally considered to be a type 1 hypervisor. If there is an OS present, and if the VMM/hypervisor operates as a separate layer, it will be considered as a type 2 hypervisor. Once again, this concept is open to debate and there is no standard definition for this. A type 1 hypervisor directly interacts with the system hardware; it does not need any host OS. You can directly install it on a bare-metal system and make it ready to host virtual machines. Type 1 hypervisors are also called **bare-metal**, **embedded**, or **native hypervisors**. oVirt-node, VMware ESXi/vSphere, and **Red Hat Enterprise Virtualization Hypervisor** (**RHEV-H**) are examples of a type 1 Linux hypervisor. The following diagram pro-

vides an illustration of the type 1 hypervisor design concept:

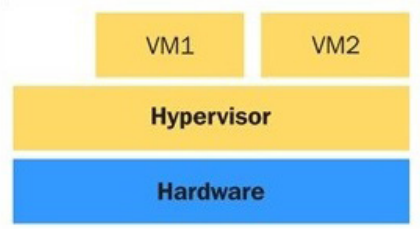

Figure 1.1 – Type 1 hypervisor design

Here are the advantages of type 1 hypervisors:

- Easy to install and configure
- Small in size; optimized to give most of the physical resources to the hosted guest (virtual machines)
- Generates less overhead as it comes with only the applications needed to run virtual machines
- More secure, because problems in one guest system do not affect the other guest systems running on the hypervisor

However, a type 1 hypervisor doesn't favor customization. Generally, there will be some restrictions when you try to install any third-party applications or drivers on it.

On the other hand, a type 2 hypervisor resides on top of the OS, allowing you to do numerous customizations. Type 2 hypervisors are also known as hosted hypervisors that are dependent on the host OS for their operations. The main advantage of type 2 hypervisors is the wide range of hardware support, because the underlying host OS controls hardware access. The following diagram provides an illustration of the type 2 hypervisor design concept:

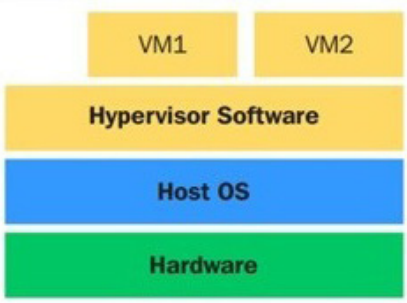

Figure 1.2 – Type 2 hypervisor design

When do we use type 1 versus type 2 hypervisors? It primarily depends on whether we already have an OS running on a server where we want to deploy virtual machines. For example, if we're already running a Linux desktop on our workstation, we're probably not going to format our workstation and install a hypervisor – it just

wouldn't make any sense. That's a good example of a type 2 hypervisor use case. Well-known type 2 hypervisors include VMware Player, Workstation, Fusion, and Oracle VirtualBox. On the other hand, if we're specifically aiming to create a server that we're going to use to host virtual machines, then that's type 1 hypervisor territory.

Open source virtualization projects

The following table is a list of open source virtualization projects in Linux:

Project	Virtualization type	Project URL
KVM (Kernel-based Virtual Machine)	Full virtualization	http://www.linux-kvm.org/
VirtualBox	Full virtualization	https://www.virtualbox.org/
Xen	Full and paravirtualization	http://www.xenproject.org/
Lguest	Paravirtualization	http://lguest.ozlabs.org/
UML (User Mode Linux)	First	http://user-mode-linux.sourceforge.net/
Linux-VServer	First	http://www.linuxvirtualserver.org/

Figure 1.3 – Open source virtualization projects in Linux

In the upcoming sections, we will discuss Xen and KVM, which are the leading open source vir-

tualization solutions in Linux.

Xen

Xen originated at the University of Cambridge as a research project. The first public release of Xen was in 2003. Later, the leader of this project at the University of Cambridge, Ian Pratt, co-founded a company called XenSource with Simon Crosby (also from the University of Cambridge). This company started to develop the project in an open source fashion. On 15 April 2013, the Xen project was moved to the Linux Foundation as a collaborative project. The Linux Foundation launched a new trademark for the Xen Project to differentiate the project from any commercial use of the older Xen trademark. More details about this can be found at **https://xenproject.org/**.

The Xen hypervisor has been ported to a number of processor families, such as Intel IA-32/64, x86_64, PowerPC, ARM, MIPS, and so on.

The core concept of Xen has four main building blocks:

- **Xen hypervisor**: The integral part of Xen that handles intercommunication between the physical hardware and virtual machine(s). It handles all interrupts, times, CPU and memory requests, and hardware interaction.
- **Dom0**: Xen's control domain, which controls a virtual machine's environment. The main part of it is called QEMU, a piece of software that emulates a regular computer system by doing binary translation to emulate a CPU.
- **Management utilities**: Command-line utilities and GUI utilities that we use to manage the overall Xen environment.
- **Virtual machines** (unprivileged domains, DomU): Guests that we're running on Xen.

As shown in the following diagram, Dom0 is a completely separate entity that controls the other virtual machines, while all the other are happily stacked next to each other using system resources provided by the hypervisor:

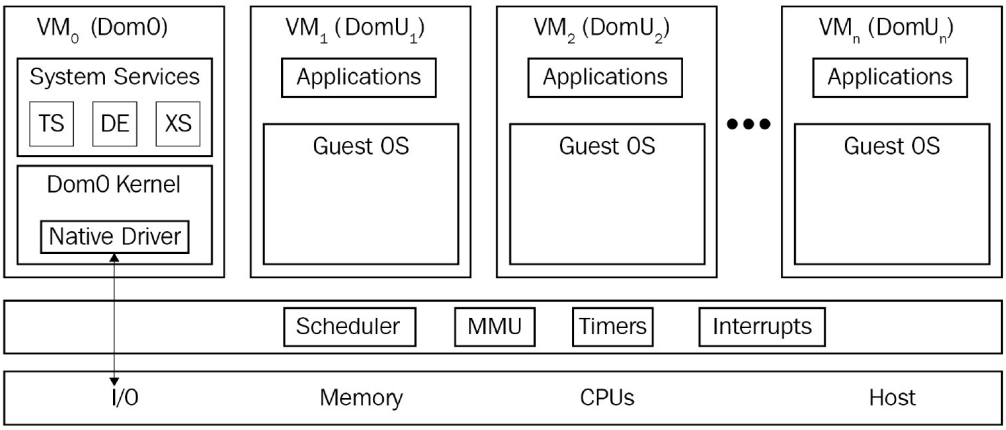

Figure 1.4 – Xen

Some management tools that we're going to mention a bit later in this book are actually capable of working with Xen virtual machines as well. For example, the **`virsh`** command can be easily used to connect to and manage Xen hosts. On the other hand, oVirt was designed around KVM virtualization and that would definitely not be the preferred solution to manage your Xen-based environment.

KVM

KVM represents the latest generation of open source virtualization. The goal of the project was to create a modern hypervisor that builds on the experience of previous generations of technolo-

gies and leverages the modern hardware available today (VT-x, AMD-V, and so on).

KVM simply turns the Linux kernel into a hypervisor when you install the KVM kernel module. However, as the standard Linux kernel is the hypervisor, it benefits from the changes that were made to the standard kernel (memory support, scheduler, and so on). Optimizations for these Linux components, such as the scheduler in the 3.1 kernel, improvement to nested virtualization in 4.20+ kernels, new features for mitigation of Spectre attacks, support for AMD Secure Encrypted Virtualization, Intel iGPU passthrough in 4/5.x kernels, and so on benefit both the hypervisor (the host OS) and the Linux guest OSes. For I/O emulations, KVM uses a userland software, QEMU; this is a userland program that does hardware emulation.

QEMU emulates the processor and a long list of peripheral devices such as the disk, network, VGA, PCI, USB, serial/parallel ports, and so on to build a complete piece of virtual hardware that

the guest OS can be installed on. This emulation is powered by KVM.

What Linux virtualization offers you in the cloud

The cloud is *the buzzword* that's been a part of almost all IT-related discussions in the past 10 or so years. If we take a look at the history of cloud, we'll probably realize the fact that Amazon was the first key player in the cloud market, with the release of **Amazon Web Services** (**AWS**) and **Amazon Elastic Compute Cloud** (**EC2**) in 2006. Google Cloud Platform was released in 2008, and Microsoft Azure was released in 2010. In terms of the **Infrastructure-as-a-Service** (**IaaS**) cloud models, these are the biggest IaaS cloud providers now, although there are others (IBM Cloud, VMware Cloud on AWS, Oracle Cloud, and Alibaba Cloud, to name a few). If you go through this list, you'll soon realize that most of these cloud platforms are based on Linux (just as an

example, Amazon uses Xen and KVM, while Google Cloud uses KVM virtualization).

Currently, there are three main open source cloud projects that use Linux virtualization to build IaaS solutions for the private and/or hybrid cloud:

- **OpenStack**: A fully open source cloud OS that consists of several open source sub projects that provide all the building blocks to create an IaaS cloud. KVM (Linux virtualization) is the most used (and best-supported) hypervisor in OpenStack deployments. It's governed by the vendor-agnostic OpenStack Foundation. How to build an OpenStack cloud using KVM will be explained in detail in ***Chapter 12***, *Scaling out KVM with OpenStack*
- **CloudStack** This is another open source **Apache Software Foundation** (**ASF**)-controlled cloud project used to build and manage highly scalable multitenant IaaS clouds and is fully compatible with EC2/S3 APIs. Although it supports all top-level Linux hypervisors, most

CloudStack users choose Xen as it is tightly integrated with CloudStack.

- **Eucalyptus**: This is an AWS-compatible private cloud software for organizations to use in order to reduce their public cloud cost and regain control over security and performance. It supports both Xen and KVM as a computing resources provider.

There are other important questions to consider when discussing OpenStack beyond the technical bits and pieces that we've discussed so far in this chapter. One of the most important concepts in IT today is actually being able to run an environment (purely virtualized one, or a cloud environment) that includes various types of solutions (such as virtualization solutions) by using some kind of management layer that's capable of working with different solutions at the same time. Let's take OpenStack as an example of this. If you go through the OpenStack documentation, you'll soon realize that OpenStack supports 10+ different virtualization solutions, including the following:

- KVM
- Xen (via libvirt)
- LXC (Linux containers)
- Microsoft Hyper-V
- VMware ESXi
- Citrix XenServer
- **User Mode Linux (UML)**
- PowerVM (IBM Power 5-9 platform)
- Virtuozzo (hyperconverged solution that can use virtual machines, storage, and containers)
- z/VM (virtualization solution for IBM Z and IBM LinuxONE servers)

That brings us to the multi-cloud environments that could span different CPU architectures, different hypervisors, and other technologies such as hypervisors – all under the same management toolset. This is just one thing that you can do with OpenStack. We'll get back to the subject of OpenStack later in this book, specifically in ***Chapter 12***, *Scaling Out KVM with OpenStack*.

Summary

In this chapter, we covered the basics of virtualization and its different types. Keeping in mind the importance of virtualization in today's large-scale IT world is beneficial as it's good to know how these concepts can be tied together to create a bigger picture – large, virtualized environments and cloud environments. Cloud-based technologies will be covered later in much greater detail – treat what we've mentioned so far as a starter; the main course is still to come. But the next chapter belongs to the main star of our book – the KVM hypervisor and its related utilities.

Questions

1. Which types of hypervisors exist?
2. What are containers?
3. What is container-based virtualization?
4. What is OpenStack?

Further reading

Please refer to the following links for more information regarding what was covered in this chapter:

- What is KVM?: **https://www.redhat.com/en/topics/virtualization/what-is-KVM**
- KVM hypervisors: **https://www.linux-kvm.org/page/Main_Page**
- OpenStack Platform: **https://www.openstack.org**
- Xen Project: **https://xenproject.org/**