# 1
# Understanding Linux Virtualization

This chapter provides the reader with an insight into the prevailing technologies in Linux virtualization and their advantage over others. There are a total of 14 chapters in this book, which are lined up to cover all the important aspects of KVM virtualization, starting from KVM internals and advanced topics such as software defined networking, performance tuning, and optimization, to physical to virtual migration.

In this chapter, we will cover the following topics:

- Linux virtualization and its basic concepts
- Why you should use Linux virtualization
- Hypervisor/VMM
- What Linux virtualization offers you in the cloud
- Public and private clouds

Before you start, check out the homepage of the book `http://bit.ly/mkvmvirt` to see the new updates, tips and version changes.

# What is virtualization?

In philosophy, virtual means "something that is not real". In computer science, virtual means "a hardware environment that is not real". Here, we duplicate the functions of physical hardware and present them to an operating system. The technology that is used to create this environment can be called **virtualization technology**, in short, virtualization. The physical system that runs the virtualization software (hypervisor or Virtual Machine Monitor) is called a host and the virtual machines installed on top of the hypervisor are called guests.

# Why should I use Linux virtualization?

Virtualization first appeared in Linux in the form of **User-mode Linux** (**UML**) and it started the revolution required to bring Linux into the virtualization race. Today, there is a wide array of virtualization options available in Linux to convert a single computer into multiple ones. Popular Linux virtualization solutions include KVM, Xen, QEMU, and VirtualBox. In this book, we will be focusing on KVM virtualization.

Openness, flexibility, and performance are some of the major factors that attract users to Linux virtualization. Just like any other open source software, virtualization software in Linux is developed in a collaborative manner; this indirectly brings users the advantages of the open source model. For example, compared to closed source, open source receives wider input from the community and indirectly helps reduce research and development costs, improves efficiency, and performance and productivity. The open source model always encourages innovation. The following are some of the other features that open source provides:

- User-driven solutions for real problems
- Support from the community and a user base who help fellow users to solve problems
- Provides choice of infrastructure
- Control of data and security, as the code is freely available to read, understand, and modify when required
- Avoid lock-in flexibility to migrate the entire load with comparable product and stay free from vendor lock-in

# Types of virtualization

Simply put, virtualization is the process of virtualizing something such as hardware, network, storage, application, access, and so on. Thus, virtualization can happen to any of the components.

Refer to the *Advantages of virtualization* section for more details on different possibilities in virtualization.

For example:

- **SDN** or **Software-Defined Networking**, `https://en.wikipedia.org/wiki/Software-defined_networking`. These techniques are examples of network virtualization, `https://en.wikipedia.org/wiki/Network_virtualization`.
- **Software Defined Storage** (**SDS**), `https://en.wikipedia.org/wiki/Software-defined_storage`. This is part of storage virtualization, `https://en.wikipedia.org/wiki/Storage_virtualization`.
- The application streaming, remote desktop service, and desktop virtualization techniques fall into the category of application virtualization, `https://en.wikipedia.org/wiki/Application_virtualization`.

However, in the context of our book, we will discuss virtualization mainly in terms of software (hypervisor-based) virtualization. From this angle, virtualization is the process of hiding the underlying physical hardware so that it can be shared and used by multiple operating systems. This is also known as platform virtualization. In short, this action introduces a layer called a **hypervisor/VMM** between the underlying hardware and the operating systems running on top of it. The operating system running on top of the hypervisor is called the guest or virtual machine.

# Advantages of virtualization

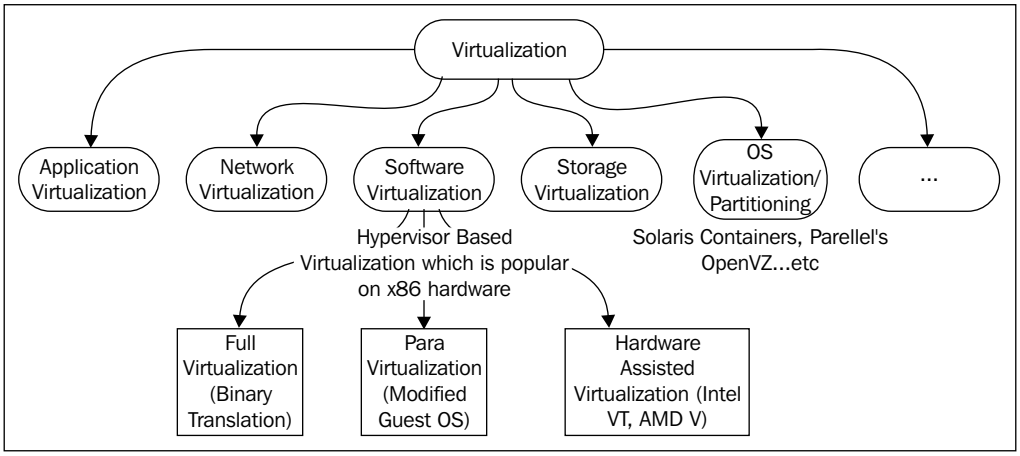
Let's discuss some of the advantages of virtualization:

- **Server consolidation**: It is well understood that virtualization helps in saving power and having a smaller energy footprint. Server consolidation with virtualization will also reduce the overall footprint of the entire data center. Virtualization reduces the number of physical or bare metal servers, reducing networking stack components and other physical components, such as racks. Ultimately, this leads to reduced floor space, power savings, and so on. This can save you more money and also help with energy utilization. Does it also ensure increased hardware utilization? Yes, it does. We can provision virtual machines with the exact amount of CPU, memory, and storage resources that they need and this will in turn make sure that hardware utilization is increased.

- **Service isolation**: Suppose no virtualization exists; in this scenario, what's the solution to achieve service isolation? Isn't it that we need to run one application per physical server? Yes, this can make sure that we achieve service isolation; however, will it not cause physical server sprawl, underutilized servers, and increased costs? Without any doubt, I can say that it does. The server virtualization helps application isolation and also removes application compatibility issues by consolidating many of these virtual machines across fewer physical servers. In short, service isolation technique this brings the advantage of simplified administration of services.
- **Faster server provisioning**: Provisioning a bare metal system will consume some time, even if we have some automated process in the path. But in case of virtualization, you can spawn a virtual machine from prebuilt images (templates) or from snapshots. It's that quick, as you can imagine. Also, you really don't have to worry about physical resource configuration, such as "network stack", which comes as a burden for physical or bare metal server provisioning.
- **Disaster recovery**: Disaster recovery becomes really easy when you have a virtualized data center. Virtualization allows you to take up-to-date snapshots of virtual machines. These snapshots can be quickly redeployed so you can reach to a state where everything was working fine. Also, virtualization offers features such as online and offline VM migration techniques so that you can always move those virtual machines elsewhere in your data center. This flexibility assists with a better disaster recovery plan that's easier to enact and has a higher success rate.
- **Dynamic load balancing**: Well, this depends on the policies you set. As server workloads vary, virtualization provides the ability for virtual machines, which are overutilizing the resources of a server, to be moved (live migration) to underutilized servers, based on the policies you set. Most of the virtualization solutions come with such policies for the user. This dynamic load balancing creates efficient utilization of server resources.

- **Faster development and test environment**: Think of this, if you want to test environment in a temporary manner. It's really difficult to deploy it in physical servers, isn't it? Also, it won't be of much worth if you set up this environment in a temporary manner. But it's really easy to set up a development or test environment with virtualization. Using a guest operating system/VM enables rapid deployment by isolating the application in a known and controlled environment. It also eliminates lots of unknown factors, such as mixed libraries, caused by numerous installs. Especially, if it's a development or test environment, we can expect severe crashes due to the experiments happening with the setup. It then requires hours of reinstallation, if we are on physical or bare metal servers. However, in case of VMs, it's all about simply copying a virtual image and trying again.
- **Improved system reliability and security**: A virtualization solution adds a layer of abstraction between the virtual machine and the underlying physical hardware. It's common for data on your physical hard disk to get corrupted due to some reason and affect the entire server. However, if it is stored in a virtual machine hard disk, the physical hard disk in the host system will be intact, and there's no need to worry about replacing the virtual hard disk. In any other instance, virtualization can prevent system crashes due to memory corruption caused by software such as the device drivers. The admin has the privilege to configure virtual machines in an independent and isolated environment. This sandbox deployment of virtual machines can give more security to the infrastructure because the admin has the flexibility to choose the configuration that is best suited for this setup. If the admin decides that a particular VM doesn't need access to the Internet or to other production networks, the virtual machine can be easily configured behind the network hop with a completely isolated network configuration and restrict the access to the rest of the world. This helps reduce risks caused by the infection of a single system that then affects numerous production computers or virtual machines.

- **OS independence or a reduced hardware vendor lock-in**: Virtualization is all about creating an abstraction layer between the underlying hardware and presenting a virtual hardware to the guest operating systems running on top of the stack. Virtualization eliminates the hardware vendor lock-in, doesn't it? That being said, with virtualization the setup has to be tied down to one particular vendor/platform/server, especially when the virtual machines don't really care about the hardware they run on. Thus, data center admins have a lot more flexibility when it comes to the server equipment they can choose from. In short, the advantage of virtualization technology is its hardware independence and encapsulation. These features enhance availability and business continuity. One of the nice things about virtualization is the abstraction between software and hardware.

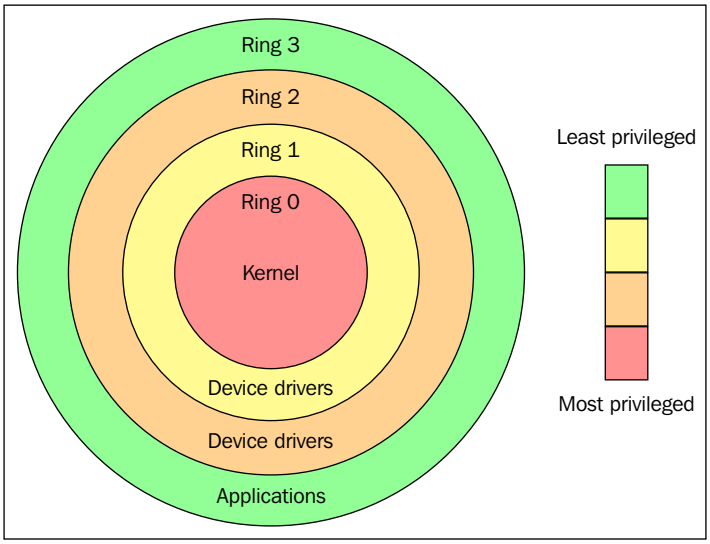


As we discussed in the preceding section, even though virtualization can be achieved in different areas, I would like to talk more about operating system virtualization and software virtualization.

# Operating system virtualization/ partitioning

The operating system virtualization technique allows the same physical host to serve different workloads and isolate each of the workloads. Please note that these workloads operate independently on the same OS. This allows a physical server to run multiple isolated operating system instances, called **containers**. There is nothing wrong if we call it container-based virtualization. The advantage of this type of virtualization is that the host operating system does not need to emulate system call interfaces for operating systems that differ from it. Since the mentioned interfaces are not present, alternative operating systems cannot be virtualized or accommodated in this type of virtualization. This is a common and well-understood limitation of this type of virtualization. Solaris containers, FreeBSD jails, and Parallel's OpenVZ fall into this category of virtualization. While using this approach, all of the workloads run on a single system. The process isolation and resource management is provided by the kernel. Even though all the virtual machines/containers are running under the same kernel, they have their own file system, processes, memory, devices, and so on. From another angle, a mixture of Windows, Unix, and Linux workloads on the same physical host are not a part of this type of virtualization. The limitations of this technology are outweighed by the benefits to performance and efficiency, because one operating system is supporting all the virtual environments. Furthermore, switching from one partition to another is very fast.

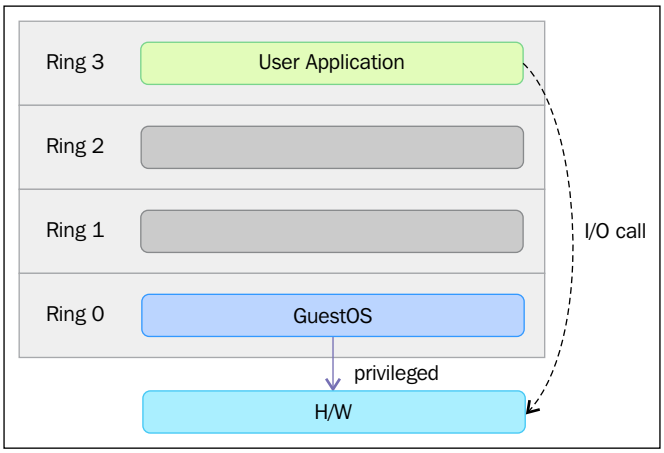
Before we discuss virtualization further and dive into the next type of virtualization, (hypervisor-based/software virtualization) it would be useful to be aware of some jargon in computer science. That being said, let's start with something called "protection rings". In computer science, various hierarchical protection domains/ privileged rings exist. These are the mechanisms that protect data or faults based on the security enforced when accessing the resources in a computer system. These protection domains contribute to the security of a computer system.




Source: `https://en.wikipedia.org/wiki/Protection_ring`

As shown in the preceding figure, the protection rings are numbered from the most privileged to the least privileged. Ring 0 is the level with the most privileges and it interacts directly with physical hardware, such as the CPU and memory. The resources, such as memory, I/O ports, and CPU instructions are protected via these privileged rings. Ring 1 and 2 are mostly unused. Most of the general purpose systems use only two rings, even if the hardware they run on provides more CPU modes (`https://en.m.wikipedia.org/wiki/CPU_modes`) than that. The main two CPU modes are the kernel mode and user mode. From an operating system's point of view, Ring 0 is called the kernel mode/supervisor mode and Ring 3 is the user mode. As you assumed, applications run in Ring 3.

Operating systems, such as Linux and Windows use supervisor/kernel and user mode. A user mode can do almost nothing to the outside world without calling on the kernel or without its help, due to its restricted access to memory, CPU, and I/O ports. The kernels can run in privileged mode, which means that they can run on ring 0. To perform specialized functions, the user mode code (all the applications run in ring 3) must perform a system call (`https://en.m.wikipedia.org/wiki/System_call`) to the supervisor mode or even to the kernel space, where a trusted code of the operating system will perform the needed task and return the execution back to the user space. In short, the operating system runs in ring 0 in a normal environment. It needs the most privileged level to do resource management and provide access to the hardware. The following image explains this:

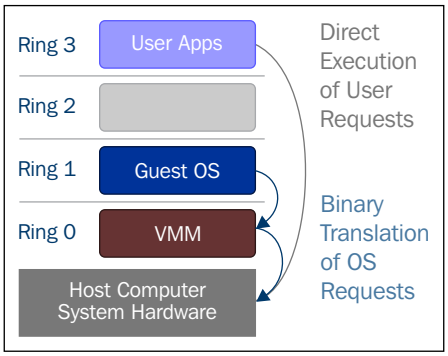



The rings above 0 run instructions in a processor mode called unprotected. The hypervisor/**Virtual Machine Monitor** (**VMM**) needs to access the memory, CPU, and I/O devices of the host. Since, only the code running in ring 0 is allowed to perform these operations, it needs to run in the most privileged ring, which is Ring 0, and has to be placed next to the kernel. Without specific hardware virtualization support, the hypervisor or VMM runs in ring 0; this basically blocks the virtual machine's operating system in ring-0. So the VM's operating system has to reside in Ring 1. An operating system installed in a VM is also expected to access all the resources as it's unaware of the virtualization layer; to achieve this, it has to run in Ring 0 similar to the VMM. Due to the fact that only one kernel can run in Ring 0 at a time, the guest operating systems have to run in another ring with fewer privileges or have to be modified to run in user mode.

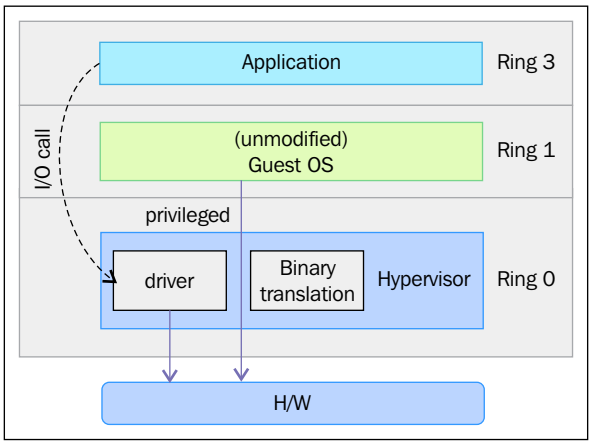
This has resulted in the introduction of a couple of virtualization methods called full virtualization and paravirtualization, which we will discuss in the following sections.

# Full virtualization

In full virtualization, privileged instructions are emulated to overcome the limitations arising from the guest operating system running in ring 1 and VMM runnning in Ring 0. Full virtualization was implemented in first-generation x86 VMMs. It relies on techniques, such as binary translation (`https://en.wikipedia.org/wiki/Binary_translation`) to trap and virtualize the execution of certain sensitive and non-virtualizable instructions. This being said, in binary translation, some system calls are interpreted and dynamically rewritten. Following diagram depicts how Guest OS access the host computer hardware through Ring 1 for privileged instructions and how un-privileged instructions are executed without the involvement of Ring 1:




With this approach, the critical instructions are discovered (statically or dynamically at runtime) and replaced with traps into the VMM that are to be emulated in software. A binary translation can incur a large performance overhead in comparison to a virtual machine running on natively virtualized architectures.

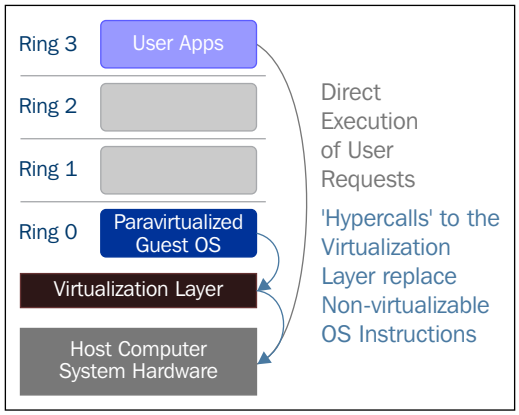


However, as shown in the preceding image, when we use full virtualization we can use the unmodified guest operating systems. This means that we don't have to alter the guest kernel to run on a VMM. When the guest kernel executes privileged operations, the VMM provides the CPU emulation to handle and modify the protected CPU operations, but as mentioned earlier, this causes performance overhead compared to the other mode of virtualization, called paravirtualization.

# Paravirtualization

In paravirtualization, the guest operating system needs to be modified in order to allow those instructions to access Ring 0. In other words, the operating system needs to be modified to communicate between the VMM/hypervisor and the guest through the "backend" (hypercalls) path.

Please note that we can also call VMM a hypervisor.



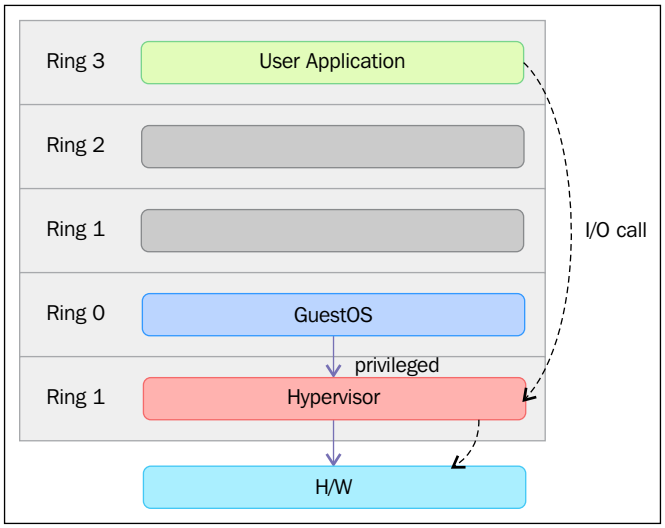

Paravirtualization (`https://en.wikipedia.org/wiki/Paravirtualization`) is a technique in which the hypervisor provides an API and the OS of the guest virtual machine calls that API which require host operating system modifications. Privileged instruction calls are exchanged with the API functions provided by the VMM. In this case, the modified guest operating system can run in ring 0.

As you can see, under this technique the guest kernel is modified to run on the VMM. In other terms, the guest kernel knows that it's been virtualized. The privileged instructions/operations that are supposed to run in ring 0 have been replaced with calls known as hypercalls, which talk to the VMM. The hypercalls invoke the VMM to perform the task on behalf of the guest kernel. As the guest kernel has the ability to communicate directly with the VMM via hypercalls, this technique results in greater performance compared to full virtualization. However, This requires specialized guest kernel which is aware of para virtualization technique and come with needed software support.

# Hardware assisted virtualization

Intel and AMD realized that full virtualization and paravirtualization are the major challenges of virtualization on the x86 architecture (as the scope of this book is limited to x86 architecture, we will mainly discuss the evolution of this architecture here) due to the performance overhead and complexity in designing and maintaining the solution. Intel and AMD independently created new processor extensions of the x86 architecture, called Intel VT-x and AMD-V respectively. On the Itanium architecture, hardware-assisted virtualization is known as VT-i. Hardware assisted virtualization is a platform virtualization method designed to efficiently use full virtualization with the hardware capabilities. Various vendors call this technology by different names, including accelerated virtualization, hardware virtual machine, and native virtualization.

For better support of for virtualization, Intel and AMD introduced **Virtualization Technology** (**VT**) and **Secure Virtual Machine** (**SVM**), respectively, as extensions of the IA-32 instruction set. These extensions allow the VMM/hypervisor to run a guest OS that expects to run in kernel mode, in lower privileged rings. Hardware assisted virtualization not only proposes new instructions, but also introduces a new privileged access level, called ring -1, where the hypervisor/VMM can run. Hence, guest virtual machines can run in ring 0. With hardware-assisted virtualization, the operating system has direct access to resources without any emulation or OS modification. The hypervisor or VMM can now run at the newly introduced privilege level, Ring -1, with the guest operating systems running on Ring 0. Also, with hardware assisted virtualization, the VMM/hypervisor is relaxed and needs to perform less work compared to the other techniques mentioned, which reduces the performance overhead.




In simple terms, this virtualization-aware hardware provides the support to build the VMM and also ensures the isolation of a guest operating system. This helps to achieve better performance and avoid the complexity of designing a virtualization solution. Modern virtualization techniques make use of this feature to provide virtualization. One example is KVM, which we are going to discuss in detail in the scope of this book.

# Introducing VMM/hypervisor

As its name suggests, the VMM or hypervisor is a piece of software that is responsible for monitoring and controlling virtual machines or guest operating systems. The hypervisor/VMM is responsible for ensuring different virtualization management tasks, such as providing virtual hardware, VM life cycle management, migrating of VMs, 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. The multiple guest systems can be on the same operating system or different ones. For example, there can be multiple Linux guest systems running as guests on the same physical system. The VMM is responsible to allocate the resources requested by these guest operating systems. The system hardware, such as the processor, memory, and so on has to be allocated to these guest operating systems according to their configuration, and VMM can take care of this task. Due to this, VMM is a critical component in a virtualization environment.

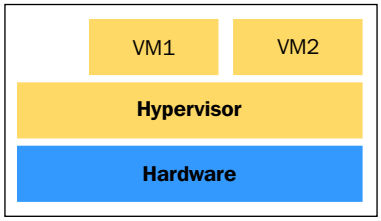
Depending on the location of the VMM/hypervisor and where it's placed, it is categorized either as type 1 or type 2.

# 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 operating system 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 operating system 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 operating system. 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 is an example of a Type 1 Linux hypervisor. The following figure provides an illustration of the Type 1 hypervisor design concept:

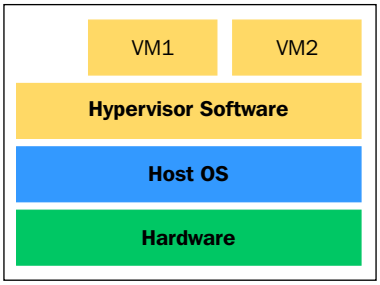



Here are the advantages of Type 1 hypervisors:

- 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, you will not be allowed to install any third party applications or drivers on it.

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




Deciding on the type of hypervisor to use mainly depends on the infrastructure of where you are going to deploy virtualization.

Also, there is a concept that Type 1 hypervisors perform better when compared to Type 2 hypervisors, as they are placed directly on top of the hardware. It does not make much sense to evaluate performance without a formal definition of Type 1 and Type 2 hypervisors.

# 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) | | `http://user-mode-linux.sourceforge.net/` |
| Linux-VServer | | `http://www.linux-vserver.org/Welcome_to_Linux-VServer.org` |

In upcoming sections, we will discuss Xen and KVM, which are the leading open source virtualization solutions in Linux.

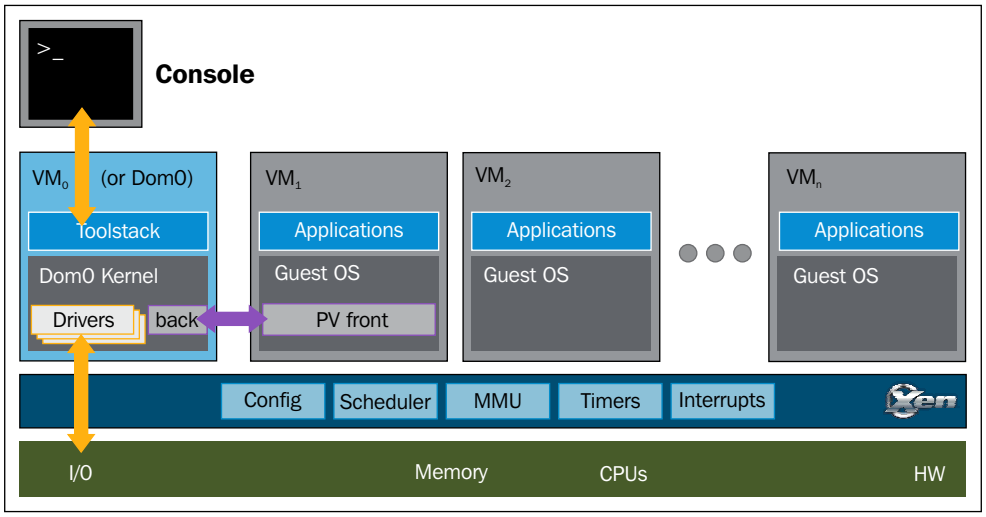
# Xen

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 of 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 `xenproject.org` website.

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

Xen can operate on both para virtualization and **Hardware-assisted or Full Virtualization** (**HVM**), which allow unmodified guests. A Xen hypervisor runs guest operating systems called **Domains**. There are mainly two types of domains in Xen:

- Dom 0
- Dom U




Source: `http://www.xenproject.org/`

Dom Us are the unprivileged domains or guest systems. Dom 0 is also known as the privileged domain or the special guest and has extended capabilities. The Dom Us or guest systems are controlled by Dom 0. That said Dom 0 contains the drivers for all the devices in the system. Dom 0 also contains a control stack to manage virtual machine creation, destruction, and configuration. Dom 0 also has the privilege to directly access the hardware; it can handle all the access to the system's I/O functions and can interact with the other Virtual Machines. Dom 0 sets the Dom Us, communication path with hardware devices using virtual drivers. It also exposes a control interface to the outside world, through which the system is controlled. Dom 0 is the first VM started by the system and it's a must-have domain for a Xen Project hypervisor.

If you want to know more about the Xen project, please refer to `http://wiki.xenproject.org/wiki/Xen_Overview` or `http://xenproject.org`

# Introducing KVM

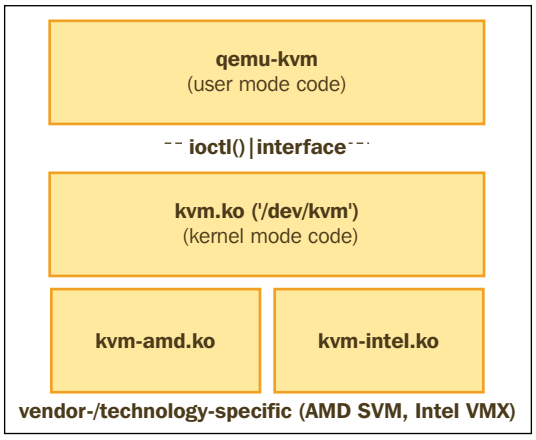
**Kernel-based Virtual Machine** (**KVM**) represents the latest generation of open source virtualization. The goal of the project was to create a modern hypervisor that builds on the experience of previous generations of technologies and leverages the modern hardware available today (VT-x, AMD-V).

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 to the standard kernel (memory support, scheduler, and so on). Optimizations to these Linux components (such as the new scheduler in the 3.1 kernel) benefit both the hypervisor (the host operating system) and the Linux guest operating systems. For I/O emulations, KVM uses a userland software, QEMU; Qemu is a userland program that does hardware emulation.

It emulates the processor and a long list of peripheral devices: disk, network, VGA, PCI, USB, serial/parallel ports, and so on to build a complete virtual hardware on which the guest operating system can be installed and this emulation is powered by KVM.

# High-level overview of KVM

The following figure gives us a high-level overview of the user mode and kernel mode components of a KVM:




A separate `qemu-kvm` process is launched for each virtual machine by `libvirtd` at the request of system management utilities, such as `virsh` and `virt-manager`. The properties of the virtual machines (number of CPUs, memory size, I/O device configuration) are defined in separate XML files, which are located in the directory `/etc/libvirt/qemu`. `libvirtd` uses the details from these XML files to derive the argument list that is passed to the `qemu-kvm` process.

Here is an example:

```
qemu     14644  9.8  6.8 6138068 1078400 ?     Sl   03:14  97:29 /usr/
bin/qemu-system-x86_64 -machine accel=kvm -name guest1 -S -machine
pc--m 5000 -realtime mlock=off -smp 4,sockets=4,cores=1,threads=1
-uuid 7a615914-ea0d-7dab-e709-0533c00b921f -no-user-config
-nodefaults -chardev socket,id=charmonitor-drive file=/dev/vms/
hypervisor2,if=none,id=drive-virtio-disk0,format=raw,cache=none,aio=na
tive -device id=net0,mac=52:54:00:5d:be:06
```

Here, an argument similar to `-m 5000` forms a 5 GB memory for the virtual machine, `--smp = 4` points to a 4 vCPU that has a topology of four vSockets with one core for each socket.

Details about what `libvirt` and `qemu` are and how they communicate each other to provide virtualization, are explained in *Chapter 2, KVM Internals*.

# What Linux virtualization offers you in the cloud

Over the years, Linux has become the first choice for developing cloud-based solutions. Many successful public cloud providers use Linux virtualization to power their underlying infrastructure. For example, Amazon, the largest IaaS cloud provider uses Xen virtualization to power their EC2 offering and similarly it's KVM that powers Digital Ocean. Digital Ocean is the third largest cloud provider in the world. Linux virtualizations are also dominating the private cloud arena.

The following is a list of open source cloud software that uses Linux virtualization for building IaaS software:

- **Openstack**: A fully open source cloud operating system, this 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 is explained in detail in *Chapter 6*, *Virtual Machine Lifecycle Management* and *Chapter 7*, *Templates and Snapshots*.
- **Cloudstack**: This is another open source **Apache Software Foundation** (**ASF**) controlled cloud project to build and manage highly-scalable multi-tenant IaaS cloud, which 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 reduce their public cloud cost and regain control over security and performance. It supports both Xen and KVM as a computing resources provider.

# Summary

In this chapter, you have learned about Linux virtualization, its advantages, and different types of virtualization methods. We also discussed the types of hypervisor and then went through the high-level architecture of Xen and KVM, and popular open source Linux virtualization technologies.

In the next chapter, we will discuss the internal workings of `libvirt`, `qemu`, and KVM, and will gain knowledge of how these components talk to each other to achieve virtualization.