1
Understanding Linux Virtualization
理解 Linux 虚拟化

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. 本章将给读者介绍有关 Linux 虚拟化的主要技术及其优势。本书一共有 14 章，涵盖了 KVM 虚拟化的所有重要方面，包括 KVM 内部实现，诸如软件定义网络，性能调优及优化等高级话题，还包括从物理载体到虚拟载体的迁移。

In this chapter, we will cover the following topics: 在本章中，我们将介绍以下几个话题：

- Linux virtualization and its basic concepts Linux 虚拟化和基本概念
- Why you should use Linux virtualization 您为什么要使用 Linux 虚拟化
- Hypervisor/VMM 虚拟机管理程序
- What Linux virtualization offers you in the cloud Linux 虚拟化在云上能为您提供什么
- Public and private clouds 公有云和私有云

> 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? | 什么是虚拟化？

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. 在哲学中，虚拟意味着“不真实的东西”。而在计算机科学中，虚拟意味着“不真实的硬件环境”。在这里，我们通过复制物理硬件的功能将其呈现给操作系统。用于创建这种环境的技术称之为虚拟化技术，简称虚拟化。运行虚拟化软件(管理程序(hypervisor)或虚拟机监视器(VMM))的物理系统称之为主机(host)，而安装在 hypervisor 之上的虚拟机则被称为客户机(guest)。

Why should I use Linux virtualization? | 为什么要使用 Linux 虚拟化？

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. 虚拟化首先以用户模式 Linux(UML)的形式出现在 Linux 中，从此 Linux 加入了虚拟化竞赛大军，历经多次变革。时至今天，将单台计算机转换成"多台计算机"，Linux 已经支持多种虚拟化选项。流行的 Linux 虚拟化解决方案包括 KVM，Xen，QEMU 和 VirtualBox。在本书中，我们将重点介绍 KVM 虚拟化。

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: 开放性，灵活性和性能是吸引用户使用 Linux 虚拟化的主要因素。就像其他开源软件一样，Linux 的虚拟化软件是以协作方式开发的；这间接为用户带来了开源模型的优势。例如：与闭源相比，开源从社区获得更广泛的输入，有助于降低研发成本，提高效率、性能和生产力。另外，开源模式总是鼓励创新。以下是开源提供的一些其他功能：

- User-driven solutions for real problems 针对实际问题的以用户为驱动的解决方案
- Support from the community and a user base who help fellow users to solve problems 来自社区的支持和来自帮助同伴用户解决问题的用户群的支持
- Provides choice of infrastructure 提供基础设施方面的选择
- Control of data and security, as the code is freely available to read, understand, and modify when required 控制数据和安全性，因为代码可以在需要时自由地阅读，理解和修改
- Avoid lock-in flexibility to migrate the entire load with comparable product and stay free from vendor lock-in 在迁移整个负载与类似产品的时候保持灵活性(避免被锁定)，并持续拥有不被供应商锁定的自由度

Types of virtualization | 虚拟化的类型

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. 简单来说，虚拟化是将硬件，网络，存储，应用程序，访问等虚拟化的过程。因此，虚拟化可能发生在任何组件中。

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

For example: 例如：

- 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 . 软件定义网络
- 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. 软件定义存储
- The application streaming, remote desktop service, and desktop virtualization techniques fall into the category of application virtualization, https://en.wikipedia.org/wiki/Application_virtualization . 应用程序流，远程桌面服务和桌面虚拟化技术属于应用程序虚拟化

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. 然而，在本书中，我们将主要讨论(基于hypervisor)软件虚拟化。从这个角度来看，虚拟化是一个将底层物理硬件隐藏起来的过程，因此它可以被多个操作系统共享和使用。这也被称为平台虚拟化。简而言之，这一操作在底层硬件和在其上运行的操作系统之间引入了一个称之为 **hypervisor/VMM** 的中间层。在管理程序之上运行的操作系统称为客户机(guest)或虚拟机。

Advantages of virtualization | 虚拟化的优势所在

Let's discuss some of the advantages of virtualization: 让我们讨论一下虚拟化的诸多优势：

- **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. 整合服务器：众所周知，虚拟化有助于节省电力和减少能源消耗。使用虚拟化的服务器整合还将减少整个数据中心的总体占用空间。虚拟化减少了物理或裸机服务器的数量，也减少了网络栈组件和其他物理组件（例如机架）。最终地，这减少了占地面积，节省了电力等。这可以帮你节省更多的钱，也有助于能源利用。它还能确保提升硬件利用率吗？ 是的，确实是的。我们可以为虚拟机提供所需的 CPU，内存和存储资源的精确数量，从而确保提高硬件利用率。

- **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. 隔离服务：假设没有虚拟化存在的情况下，实现服务隔离的解决方案是什么？是不是我们需要为每个物理服务器运行一个应用程序？ 是的，这可以确保我们实现服务隔离; 然而，它不会导致物理服务器蔓延却未能充分利用服务器从而增加成本吗？毫无疑问，(我可以说)是的。服务器虚拟化可以帮助实现应用程序的隔离，并通过跨越少量的物理服务器对这些虚拟机进行整合来消除应用程序兼容性问题。简而言之，服务隔离技术优势明显，简化了对服务的管理。

- **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. 更快地配置服务器：即使我们在路径中有一些自动化过程，配置裸机系统也将耗费一些时间。但在虚拟化的情况下，您可以从预先构建的映像(模板)或快照生成虚拟机。你可以想象这显然够快。此外，你真的不必担心物理资源配置，例如“网络栈”，这是专属于配置物理或裸机服务器才会有的负担。

- **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. 灾难恢复：当您拥有虚拟化数据中心时，灾难恢复将变得非常容易。虚拟化允许您获取虚拟机的最新快照。这些快照可以被快速地重新部署，以便达到一切都正常工作的状态。此外，虚拟化还提供了诸如联机和脱机 VM 迁移技术等功能，因此您可以随时将这些虚拟机移动到数据中心的其他位置。这种灵活性有助于做出更好的灾难恢复计划，而且更容易颁布和实施，并具有更高的成功率。

- **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

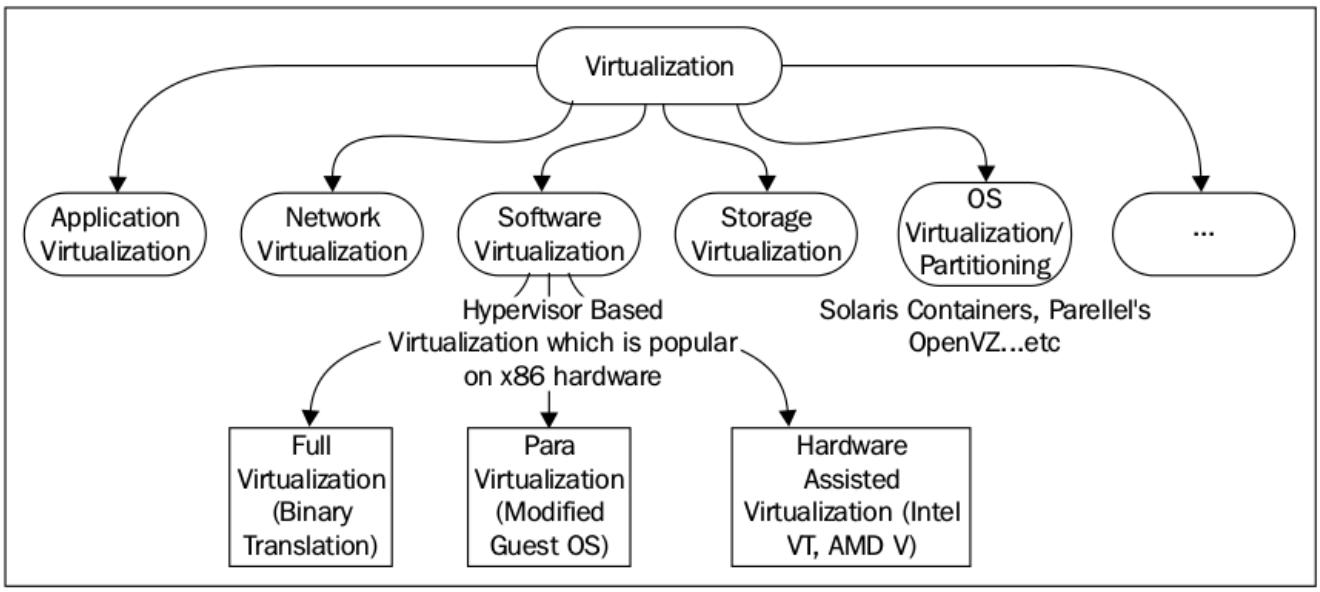
such policies for the user. This dynamic load balancing creates efficient utilization of server resources. 动态负载平衡：嗯，这取决于您设置的策略。由于服务器工作负载不同，虚拟化提供了根据您设置的策略，将被资源过度利用的服务器上的虚拟机移动(在线迁移)到资源并未被充分利用的服务器上的能力。大多数虚拟化解决方案都为用户提供了这样的策略。有了动态负载平衡，服务器资源无疑能够被有效地利用起来。

- **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. 更快地构建开发和测试环境：想想看，如果你想搭建一个临时的测试环境。它真的很难部署在物理服务器，难道不是吗？ 此外，如果您只是搭建一个临时的环境，部署在物理服务器上不是很值当。但是使用虚拟化来设置开发或测试环境真的超容易。使用客户机操作系统/VM，可以通过在已知和受控的环境中通过隔离应用程序来实现快速部署。它还消除了许多未知因素，例如由大量安装引起的混合库。特别地，如果它是一个开发或测试环境，我们可以预计到由于设置的实验将导致严重的崩溃，然后需要耗费好几个小时来重新安装，如果我们在物理或裸机服务器上搭建环境的话。但是，在 VM 的情况下，所有的一切不过是简单地复制虚拟映像，然后重试罢了。

- **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. 提高系统可靠性和安全性：虚拟化解决方案在虚拟机和底层物理硬件之间添加了一个抽象层。由于某些原因，物理硬盘上的数据被损坏，并影响到整个服务器，这是很常见的事情。然而，如果它被存储在虚拟机硬盘中，则主机系统中的物理硬盘将是完整的，并且没有必要担心将虚拟硬盘替换。在任何其他情况下，虚拟化可以防止由于诸如设备驱动程序等软件导致的内存损坏而引起的系统崩溃。管理员有权在独立和隔离的环境中配置虚拟机。这种沙箱部署虚拟机可以为基础架构提供更多的安全性，因为管理员可以灵活地选择最适合此设置的配置。如果管理员决定特定的 VM 不需要访问因特网或其他生产网络，则虚拟机可以容易地被配置在网络跳转之后，和具有完全隔离的网络配置，并且被限制访问世界其他地方。这有

助于减少由于单个系统被感染而影响到多个生产计算机或虚拟机的风险。

- **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. 保持操作系统独立性或减少了被硬件供应商锁定的可能性：虚拟化涉及在底层硬件之间创建抽象层，并向在栈顶运行的客户机操作系统呈现虚拟硬件。虚拟化消除了硬件供应商锁定，不是吗？话虽如此，使用虚拟化设置必须绑定到一个特定的供应商/平台/服务器，特别是当虚拟机不真正关心他们运行在什么硬件之上。因此，数据中心管理员在选择服务器设备方面具有更大的灵活性。总之，虚拟化技术的优势在于其硬件独立性和封装性。这些功能增强了可用性和业务连续性。虚拟化的好处之一就是提供了软件和硬件之间的一层抽象。



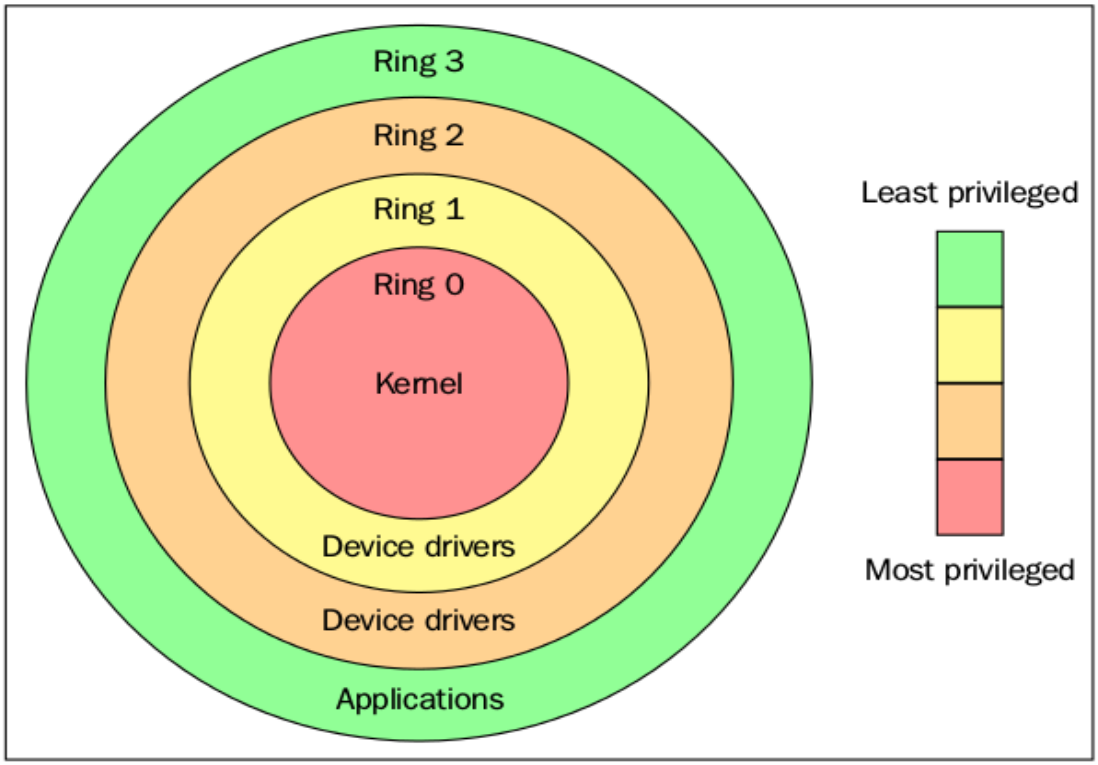
As we discussed in the preceding section, even though virtualization can be achieved in different areas, I would like to talk more about operating system virtualization and software virtualization. 正如我们在上一节中所讨论的，虚拟化可以在不同的领域得以实现。我想更多地谈一下操作系统虚拟化和软件虚拟化。

Operating system virtualization/partitioning | 操作系统虚拟化/分区

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. 操作系统虚拟化技术允许在同一个物理主机上提供不同的工作负载服务并对每一个工作负载进行隔离。请注意，这些工作负载在同一个操作系统上是独立运行的。这允许在单个物理服务器上运行多个相互隔离的操作系统实例，其被称之为容器。如果我们将其称之为基于容器的虚拟化，完全正确。这种类型的虚拟化的优点在于，主机(host)操作系统不需要为与 host 不相同的操作系统去模拟系统调用接口。因此，在这种类型的虚拟化中，不能虚拟化或容纳备选的操作系统。这种"不能"是这一类型的虚拟化的常见的和众所周知的限制。Solaris 容器 (zone)，FreeBSD jails 和 Parallel 的 OpenVZ 就属于这种类型的虚拟化。使用此方法的时候，所有工作负载都在单个系统上运行。进程隔离和资源管理由内核提供。即使所有虚拟机/容器都在同一个内核下运行，它们也有自己的文件系统，进程，内存，设备等。从另一个角度来看，在同一物理主机上的既有 Windows 又有 Unix 和 Linux 的混杂的工作负载就不是这种类型的虚拟化的一部分，其局限性优于性能和效率，因为一个操作系统就支持所有的虚拟环境。而且，从一个分区切换到另一个分区非常快。

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. 在我们进一步讨论虚拟化并深入探讨下一种类型的虚拟化(基于 hypervisor/软件虚拟化)之前，了解一下计算机科学中的一些术语是非常有用的。让我们从“保护环(ring)”开始。在计算机科学中，存在各种分级的保护域/特权环，它们是基于在访问计算机系统资源的时候强制实施的数据保护或故障安全机制。这些保护域有助于提高计算机系统的安全性。

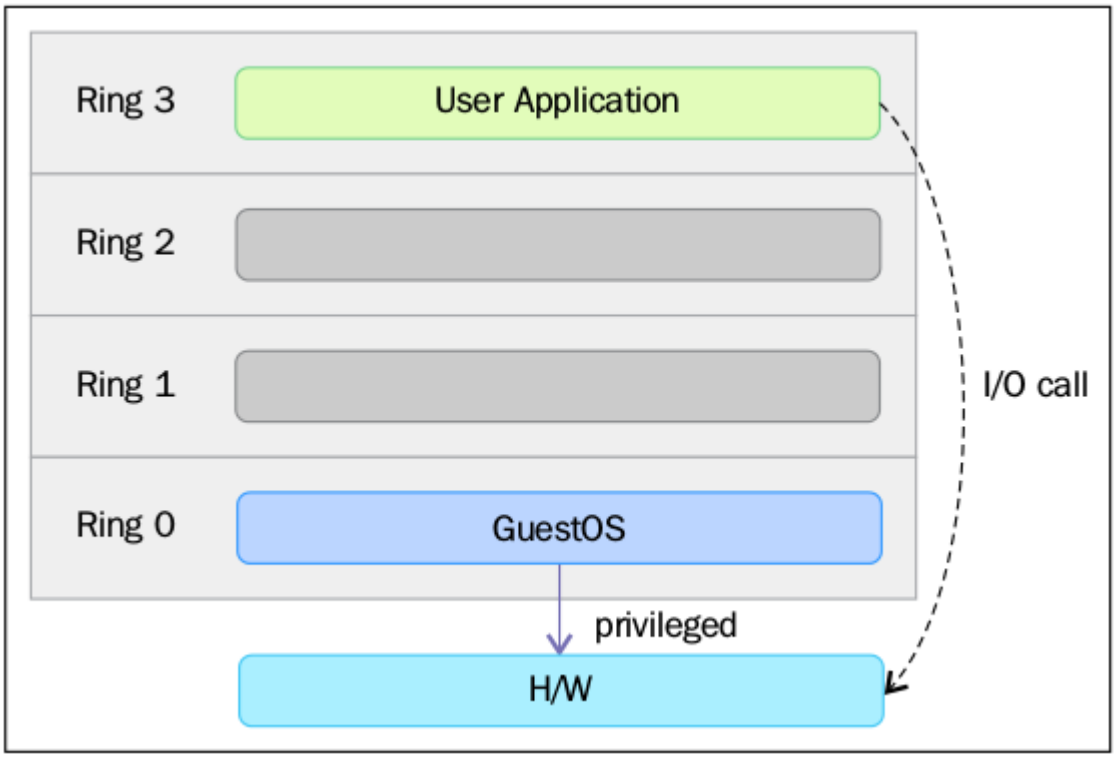


Source: https://en.wikipedia.org/wiki/Protection_ring

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. 如上图所示，保护环从最高特权一直编号到最低特权。环 0 是拥有特权最多的级别，它直接与物理硬件(如 CPU 和内存)交互。资源是通过这些特权环来保护的，例如存储器，I/O 端口和 CPU 指令。环 1 和 2 大部分未被使用。大多数通用的操作系统中只使用两个环，即使它们运行的硬件提供更多的 CPU 模式。主要的两种 CPU 模式是内核模式和用户模式。从操作系统的角度来看，环 0 被称为内核模式/supervisor 模式，环 3 是用户模式。应用程序在环 3 中运行。

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: 诸如 Linux 和 Windows 这样的操作系统，使用内核和用户两种模式。由于用户模式对内存，CPU 和 I/O 端口的访问受到限制，因此在用户模式中，几乎什么也做不了，如果不调用内核或没有内核的帮助。内核以特权模式运行，这意味着它们可以在环 0 上运行。为了执行特定的功能，用户模式代码（所有运行在环 3 的应用程序）必须执行系统调用到 supervisor 模式，甚至到内核空间，其中被操作系统信得过的代码将执行所要完成的任务，并将执行结果返回到用户空间。简而言之，在正常环境中，操作系统在环 0 中运行。它需要最高特权的级别来进行资源管理并提供对硬件的访问。下图便说明了这一点：



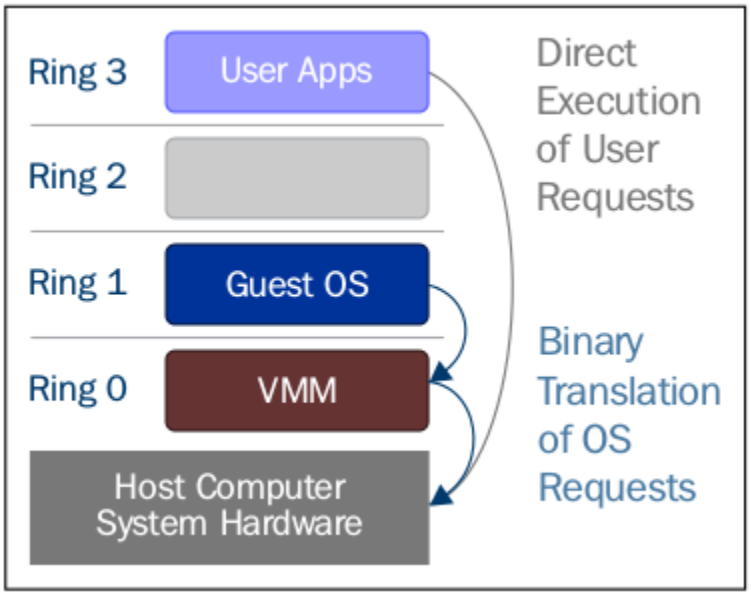
The rings above 0 run instructions in a processor mode called unprotected. The hypervisor/**Virtual Machine Monitor (VMM)** needs to access the memory, CPU, and I/O devices of the host. Since, only the code running in ring 0 is allowed to perform these operations, it needs to run in the most privileged 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. 高于 0 的环是在非保护状态的处理器模式下运行指令。hypervisor/VMM 需要访问主机的内存，CPU 和 I/O 设备。由于这些操作只有在环 0 中运行的代码中才能被允许执行，所以它需要在最高特权的环(即环 0)中运行，并且必须放置在内核旁边。如果没有特定的硬件虚拟化支持，虚拟机 hypervisor 或 VMM 运行在环 0 中，这基本上阻止了虚拟机操作系统驻留在环 0 中。因此，VM 的操作系统必须驻留在环 1 中。安装在 VM 中的操作系统也可以访问所有资源，因为它并不知道虚拟化层;为了实现这一点，它必须类似于 VMM 在环 0 中得以运行。基于每次只有一个内核可

以在环 0 中运行的事实，客户机操作系统必须在具有特权级较低的另一个环中运行，或者必须被修改从而可以在用户模式下得以运行。

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. 这引入了一些称之为全虚拟化和半(泛)虚拟化的虚拟化方法，我们将在以下部分逐一讨论。

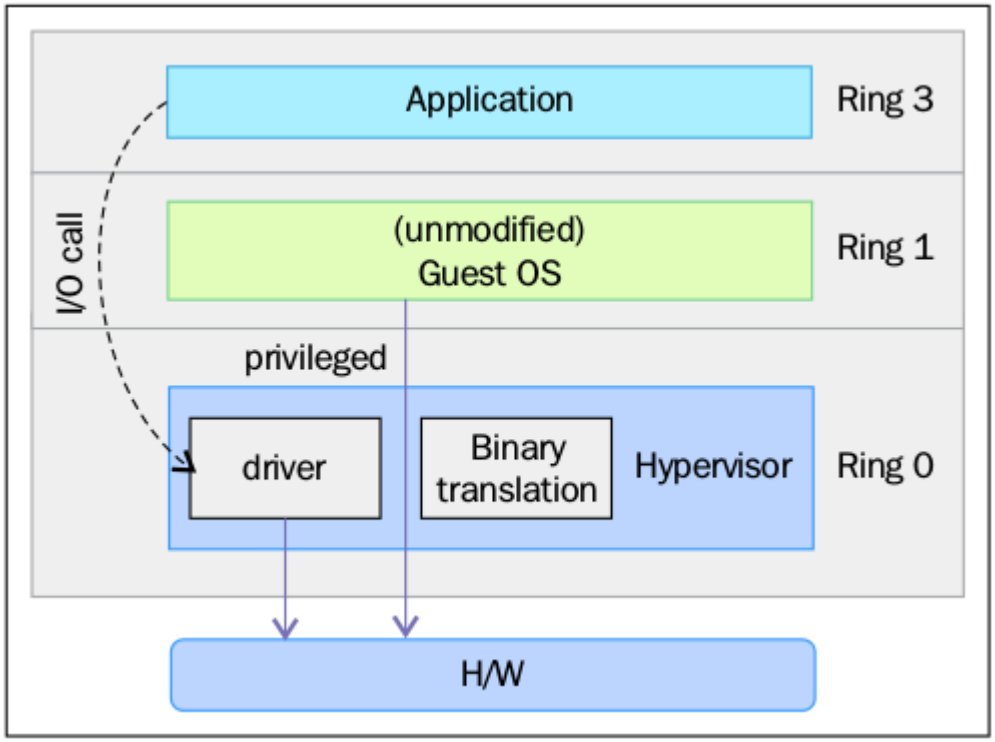
Full virtualization | 全虚拟化

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: 在全虚拟化中，为了克服在环 1 中运行的客户操作系统和在环 0 中的 VMM 运行所产生的限制，特权指令被仿真。全虚拟化在第一代 x86 VMM 中得以实现。它依赖于诸如二进制翻译之类的技术来捕获和虚拟化某些敏感和不可虚拟化的指令的执行。这就是说，在二进制翻译中，一些系统调用被动态地解释和重写。下图描述了 Guest 操作系统如何通过环 1 访问主机计算机硬件以获得特权指令，以及如何在不涉及环 1 的情况下执行非特权指令：



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. 使用这种方法，关键指令在运行的时候被静态或动态地发现，并且以软件模拟的方式被替换成 trap 陷入到 VMM 中。与在本地虚拟化体系结构上运行的虚拟机相比，二进制转换可能产生比较大的性能开销。

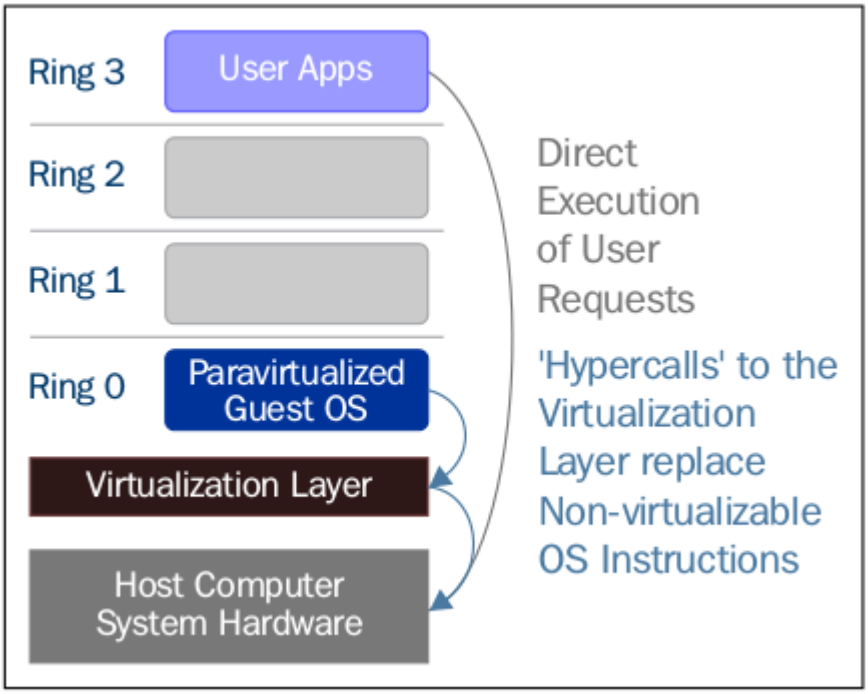


However, as shown in the preceding image, when we use full virtualization we can use the unmodified guest operating systems. This means that we don't have to alter the guest kernel to run on a VMM. When the guest kernel executes privileged operations, the VMM provides the CPU emulation to handle and modify the protected CPU operations, but as mentioned earlier, this causes performance overhead compared to the other mode of virtualization, called paravirtualization. 如前图所示，当我们使用全虚拟化时，我们可以使用未修改的客户机操作系统。这意味着我们不必更改 guest 虚拟机内核就可以在 VMM 上运行。当 guest 内核执行特权操作时，VMM 提供 CPU 仿真来处理和修改受保护的 CPU 操作，但是如前所述，与其他虚拟化模式(半/泛虚拟化)相比，性能开销较大。

Paravirtualization | 半(泛)虚拟化

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. 在半(泛)虚拟化中，guest 操作系统需要被修改以便允许指令去访问环 0。换句话说，guest 操作系统需要被修改以通过“后端”(hypercalls)路径在 VMM/hypervisor 和 guest 之间进行通信。

```
Please note that we can also call VMM a hypervisor.
```



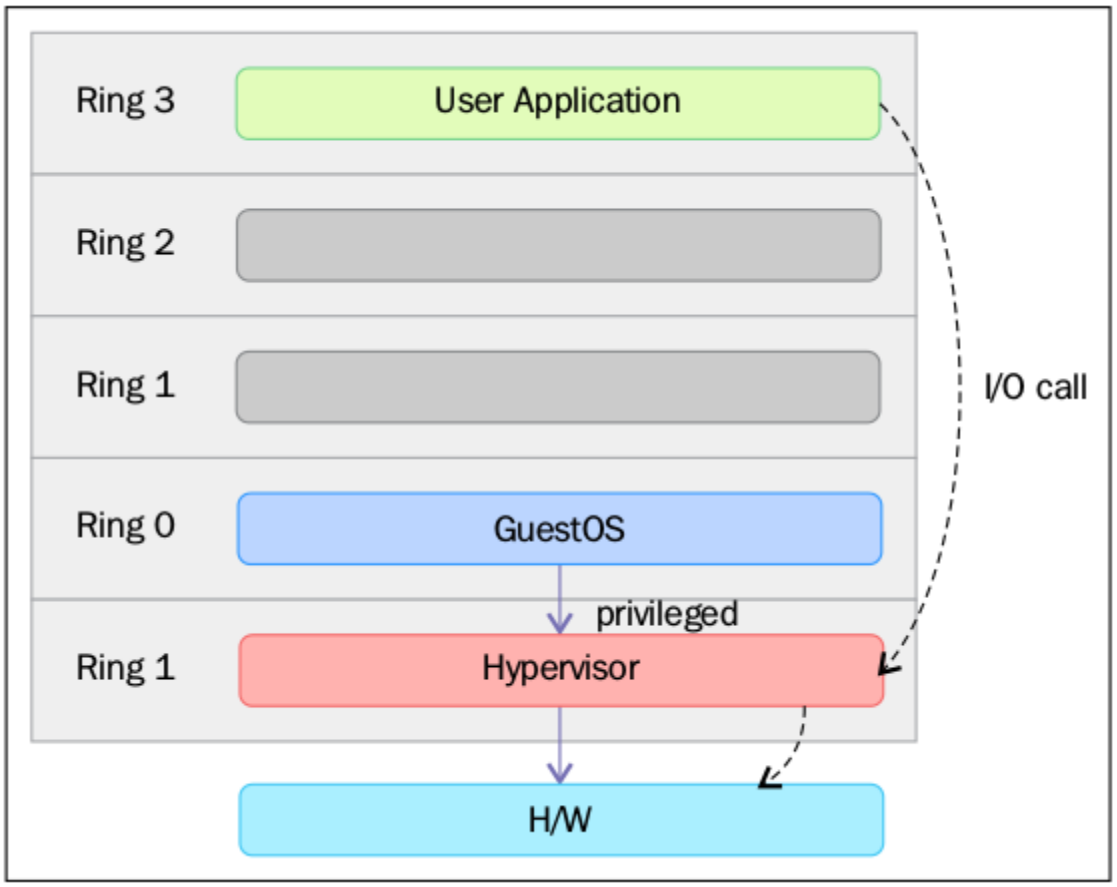
Paravirtualization (`https://en.wikipedia.org/wiki/Paravirtualization`) is a technique in which the hypervisor provides an API and the OS of the guest virtual machine calls that API which require host operating system modifications. Privileged instruction calls are exchanged with the API functions provided by the VMM. In this case, the modified guest operating system can run in ring 0. 半虚拟化是这样一种技术，其中虚拟机管理程序(hypervisor)提供一个 API，并且 guest 虚拟机的操作系统调用由 hypervisor 提供的且需要主机操作系统进行修改的 API。特权指令调用与 VMM 提供的 API 函数进行交换。在这种情况下，修改后的 guest 操作系统可以在环 0 中运行。

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. 正如你所看到的，在这种技术中，客户内核被修改为可以在 VMM 上运行的内核。换句话说，客户内核知道它已经被虚拟化。应该在环 0 中运行的特权指令/操作已经被称为超级调用(hypercall)的调用所替换，其与 VMM 通信。超级调用(hypercall)调用 VMM 代表 guest 内核执行任务。由于 guest 内核具有通过超级调用与 VMM 直接通信的能力，因此与全虚拟化相比，这种技术性能无疑更高。然而，这需要专门的 guest 内核，其知道半虚拟化技术并且具有相应的软件支持。

Hardware assisted virtualization | 硬件辅助虚拟化

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. 英特尔和 AMD 意识到，在设计和维护解决方案中，由于性能开销和复杂性的原因，全虚拟化和半虚拟化是 x86 架构虚拟化的主要挑战(本书的范围限于 x86 架构，我们将主要讨论此架构的演进)。英特尔和 AMD 独立创建了 x86 架构的新处理器扩展，分别称为 Intel VT-x 和 AMD-V。在 Itanium 架构上，硬件辅助虚拟化称为 VT-i。硬件辅助虚拟化是一种平台虚拟化方法，旨在通过硬件功能有效地使用全虚拟化。各种供应商对这种技术有不同的命名方式，例如加速虚拟化，硬件虚拟机和本地虚拟化。

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. 为了更好地支持虚拟化，英特尔和 AMD 分别引入了虚拟化技术（VT）和安全虚拟机（SVM）作为 IA-32 指令集的扩展。这些扩展允许 VMM/hypervisor 运行在且期望在较低特权环中以内核模式运行的 guest 操作系统中。硬件辅助虚拟化不仅提出新的指令，而且引入了一个新的特权访问级别，称为环-1，其中 hypervisor/VMM 可以运行在上面。因此，guest 虚拟机可以在环 0 中运行。通过硬件辅助虚拟化，操作系统可以直接访问资源，而无需任何仿真或对操作系统做修改。 hypervisor/VMM 现在可以在新引入的特权级别(环-1)上运行，其中 guest 操作系统在环 0 上运行。此外，利用硬件辅助虚拟化，VMM/hypervisor 得以放松，并且承担的工作更少了，与其他虚拟化技术(全虚拟化，半虚拟化)相比，降低了性能上的开销。



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. 简单来说，这种虚拟化感知硬件提供了对构建 VMM 的支持，并且还确保 guest 操作系统的隔离。这有助于实现更好的性能，并且避免了设计虚拟化解决方案的复杂性。现代虚拟化技术利用这一特性来提供虚拟化。一个例子是 KVM，我们将在本书的范围内给予详细的讨论。

Introducing VMM/hypervisor | VMM/hypervisor 介绍

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. 顾名思义，VMM 或 hypervisor 是一个负责监视和控制虚拟机或 guest 操作系统的软件。hypervisor/VMM 负责确保不同的虚拟化管理任务，例如提供虚拟硬件，VM 生命周期管理，VM 迁移，实时资源分配，定义虚拟机管理策略等。VMM/hypervisor 还负责有效地控制物理平台资源，例如存储器转换和 I/O 映射。虚拟化软件的一个主要优点就是能够在同一物理系统或硬件上运行多个客户机。多个客户系统可以在相同的操作系统上或不同的操作系统上运行。例如，可以有在同一物理系统上作为 guest 运行的多个 Linux 客户系统。VMM 负责分配这些 guest 操作系统请求的资源。系统硬件（如处理器，内存等）必须根据其配置分配给这些客户机操作系统，VMM 对分配任务负责。因此，VMM 是虚拟化环境中的关键组件。

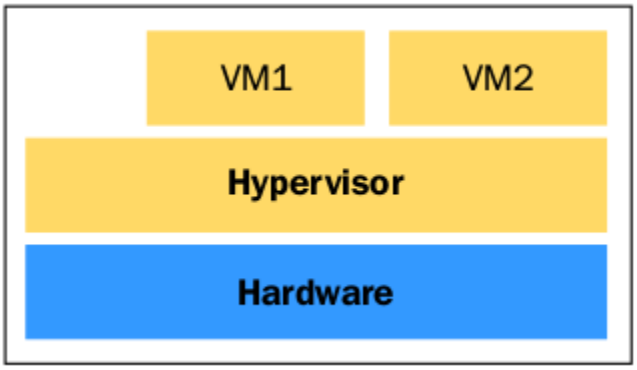
Depending on the location of the VMM/hypervisor and where it's placed, it is categorized either as type 1 or type 2. 根据 VMM/hypervisor 的位置(和被放置的位置)，它被分为两类：类型 1 或类型 2。

Type 1 and Type 2 hypervisors | 类型 1 和类型 2 的虚拟机管理程序

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. Hypervisor 主要分为两类，要么是类型 1，要么是类型 2 的 hypervisor。分类是基于它们驻留在系统中的位置，或者换句话说，底层操作系统是否存在于系统中，但是没有明确或标准的定义。如果 VMM/hypervisor 直接在硬件之上运行，其通常被认为是类型 1 的 hypervisor。如果存在于操作系统之上，并且 VMM/hypervisor 作为单独的层来操作，则它将被认为是类型 2 的 hypervisor。再说一遍，这一概念是可以有争议的，没有标准的定义。

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. 类型 1 hypervisor 直接与系统硬件交互; 它不需要任何主机操作系统。您可以直接将其安装在裸机系统上，并使其处于托管虚拟机就绪状态。类型 1 hypervisor 也称为裸机，嵌入式或本机 hypervisor。

oVirt-node is an example of a Type 1 Linux hypervisor. The following figure provides an illustration of the Type 1 hypervisor design concept: oVirt-node 是 Type 1 Linux hypervisor 的一个示例。下图提供了类型 1 hypervisor 在设计概念上的说明：

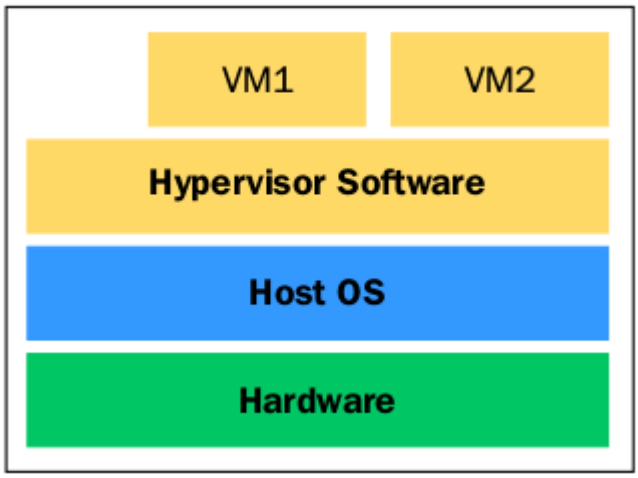


Here are the advantages of Type 1 hypervisors: 类型 1 hypervisor 的优点:

- Easy to install and configure 容易安装和配置
- Small in size, optimized to give most of the physical resources to the hosted guest (virtual machines) 体积小，经过优化，可将大部分物理资源提供给托管客户（虚拟机）
- Generates less overhead, as it comes with only the applications needed to run virtual machines 生成的开销较少，因为它只包含运行虚拟机所必需的应用程序
- More secure, because problems in one guest system do not affect the other guest systems running on the hypervisor 更安全，因为一个客户系统有问题但不会影响运行在 hypervisor 上的其他客户系统

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. 但是，类型 1 hypervisor 对自定义不"感冒"。一般来说，您将不能在其上安装任何第三方应用程序或驱动程序。

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: 另一方面，类型 2 的 hypervisor 位于操作系统之上，允许您进行大量的自定义。类型 2 hypervisor 也称为托管管理程序。类型 2 hypervisor 的操作取决于主机操作系统。类型 2 hypervisor 的主要优点是有广泛的硬件支持，因为底层主机操作系统控制着对硬件的访问。下图提供了类型 2 hypervisor 在设计概念上的说明:



Deciding on the type of hypervisor to use mainly depends on the infrastructure of where you are going to deploy virtualization. 决定使用哪种类型的 hypervisor 主要取决于你打算如何部署虚拟化的基础架构。

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. 此外，与类型 2 hypervisor 相比，类型 1 hypervisor 在概念上性能更好，因为它们是被直接放置在硬件之上。在没有类型 1 和类型 2 hypervisor 的正式定义的情况下，对它们进行性能评估意义不大。

Open source virtualization projects | 虚拟化开源项目

The following table is a list of open source virtualization projects in Linux: 下表罗列了 Linux 中的开源虚拟化项目:

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. 在接下来的部分，我们将讨论两种具有领先地位的开源的

Linux 虚拟化解决方案: Xen 和 KVM。

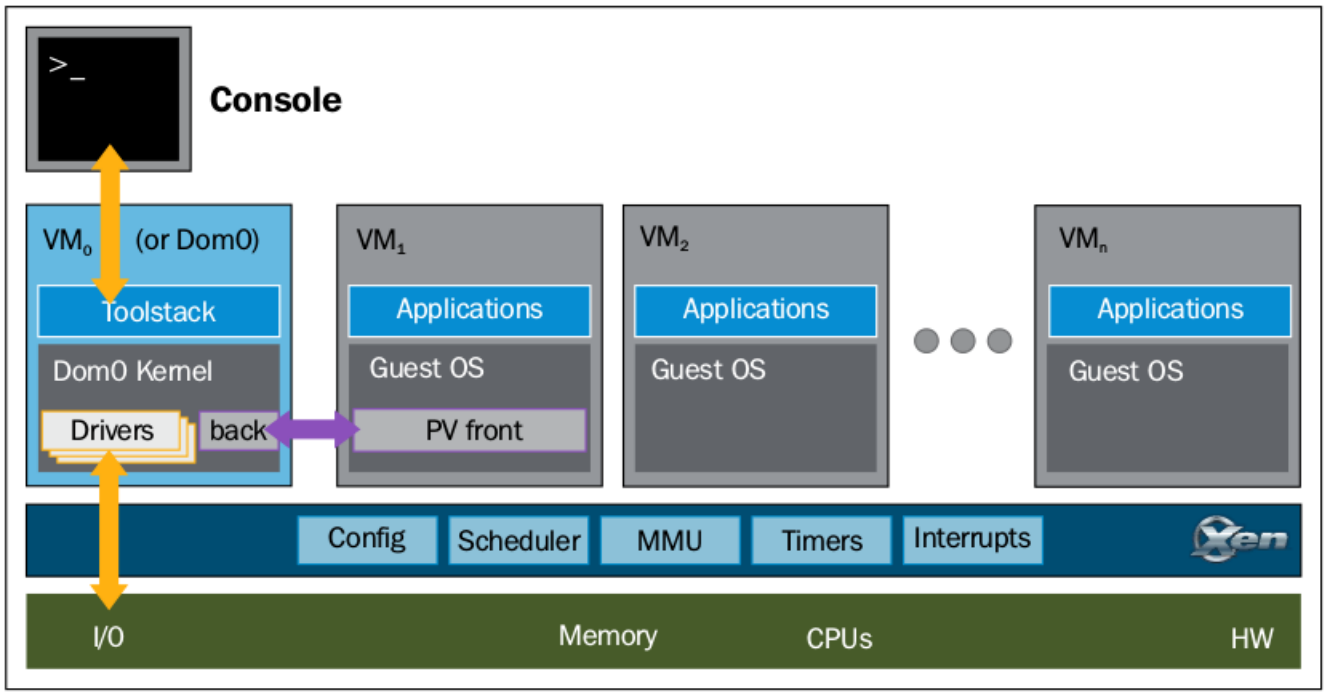
Xen

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 起源于剑桥大学一个研究项目。Xen 的第一个公开版本是在 2003 年发布。后来，剑桥大学的 Ian Prat(项目的领导者)与 Simon Crosby(也来自剑桥大学）合作创办了一家名为 XenSource 的公司。该公司开始以开源方式开发该项目。2013 年 4 月 15 日，Xen 项目作为一个合作项目移至 Linux 基金会。Linux 基金会为 Xen 项目启动了一个新的商标，以将该项目与任何商业使用的旧 Xen 商标区分开来。有关这方面的更多详细信息，请访问 xenproject.org 网站。

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 hypervisor 已经被移植到多个处理器系列上，例如，Intel IA-32/64，x86_64，PowerPC，ARM，MIPS 等。

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: Xen 操作支持半虚拟化和硬件辅助虚拟化，或者全虚拟化(HVM)(允许未修改的客户机)。Xen hypervisor 运行在叫做域(domain)的客户机操作系统上。在 Xen 中主要有两种类型的域：

- 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. Dom Us 是无特权的域或客户系统。Dom 0 也称为特权域或具有扩展功能的特殊来宾。Dom Us 或客户系统由 Dom 0 控制。Dom 0 包含系统中所有设备的驱动程序。Dom 0 还包含一个控制栈，用于管理虚拟机创建，销毁和配置。Dom 0 还有权直接访问硬件; 它可以处理对系统的 I/O 功能的所有访问，并且可以与其他虚拟机交互。Dom 0 设置 Dom Us，使用虚拟驱动程序与硬件设备的通信路径。它还暴露了到外部世界的控制接口，通过该控制接口控制系统。Dom 0 是系统启动的第一个 VM，它是 Xen Project hypervisor 必需的域。

```
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 | KVM 简介

Kernel-based Virtual Machine (KVM) represents the latest generation of open source

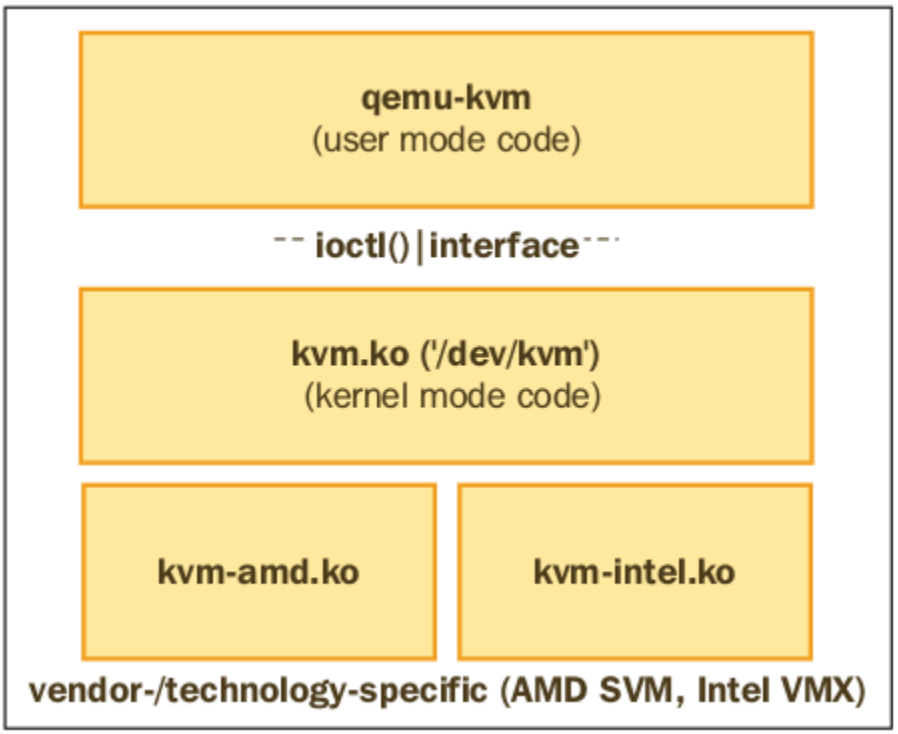
virtualization. The goal of the project was to create a modern hypervisor that builds on the experience of previous generations of technologies and leverages the modern hardware available today (VT-x, AMD-V). 基于内核的虚拟机（KVM）代表最新一代的虚拟化开源方向。该项目的目标是创建一个汲取前几代技术经验的, 充分利用现代硬件(VT-x，AMD-V)的现代虚拟机管理程序(hypervisor)。

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. 在安装 KVM 内核模块的时候，KVM 简单地将 Linux 内核转换为 hypervisor。然而，由于标准 Linux 内核就是 hypervisor，它得益于对标准内核（内存支持，调度程序等）的更改。对这些 Linux 组件（例如 3.1 内核中的新的调度器）的优化，有利于虚拟机管理程序(主机操作系统)和 Linux 客户机操作系统。对于 I/O 仿真，KVM 使用用户级软件 QEMU; 该软件是一个用户级程序，能执行硬件仿真。

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. 它通过模拟处理器和一系列的外围设备：磁盘，网络，VGA，PCI，USB，串行/并行端口等，以构建一个完整的虚拟硬件。客户操作系统可以安装在上面，并且通过 KVM 进行仿真。

High-level overview of KVM | 鸟瞰 KVM

The following figure gives us a high-level overview of the user mode and kernel mode components of a KVM: 下面给出了 KVM 的用户模式和内核模式组件的鸟瞰图：



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. libvirtd 根据系统管理实用程序（如 virsh 和 virt-manager）的请求为每个虚拟机启动单独的 qemu-kvm 进程。虚拟机的属性（CPU 个数，内存大小，I/O 设备配置）在单独的 XML 文件中定义，这些 XML 文件位于/etc/libvirt/qemu 目录中。libvirtd 使用这些 XML 文件中的详细信息来派生传递给 qemu-kvm 进程的参数列表。

Here is an example: 例如：

```
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. 这里，类似于-m 5000 的参数形成虚拟机的 5GB 内存，--smp = 4 指向具有四个 vSockets 的拓扑的 4 个 vCPU，每个 socket 具有一个 core。

Details about what libvirt and qemu are and how they communicate each other to provide virtualization, are explained in *Chapter 2, KVM Internals*. 有关 libvirt 和 qemu 是什么以及它们如何进行通信以提供虚拟化的详细信息，在第 2 章"*KVM Internals*"中有解释。

What Linux virtualization offers you in the cloud | Linux 虚拟化在云上能为您提供什么

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. 这些年来，Linux 已经成为开发基于云的解决方案的首选操作系统。许多

成功的公共云提供商使用 Linux 虚拟化来为其底层基础架构供电。例如，最大的 IaaS 云提供商亚马逊使用 Xen 虚拟化为他们的 EC2 产品供电，类似地，KVM 也为数字海洋所采用。数字海洋是世界上第三大云提供商。Linux 虚拟化也主宰着私有云领域。

The following is a list of open source cloud software that uses Linux virtualization for building IaaS software: 以下是使用 Linux 虚拟化构建 IaaS 软件的开源云软件列表：

- **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. 一个完全开源的云操作系统，它包括几个开源子项目，提供所有构建块来创建 IaaS 云。KVM（Linux 虚拟化）是 OpenStack 部署中最常用的（也是支持最好的）虚拟机管理程序。它由与供应商无关的 OpenStack 基金会控制。如何使用 KVM 构建 OpenStack 云在第 6 章“虚拟机生命周期管理”和第 7 章“模板和快照”中有详细介绍。

- **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. 这是另一个开源云项目，由 Apache 软件基金会（ASF）控制，用于构建和管理高度可扩展的多租户 IaaS 云，该云完全与 EC2/S3 API 兼容。虽然它支持所有顶级 Linux 虚拟机管理程序。大多数 Cloudstack 用户选择 Xen，因为它与 Cloudstack 紧密集成。

- **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. 这是一个与 AWS 兼容的私有云软件，用于降低组织的公共云成本并重新获得对安全性和性能的控制。它支持 Xen 和 KVM 作为计算资源提供者。

Summary | 本章总结

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. 在本章中，您了解了 Linux 虚拟化以及它的优点，和不同类型的虚拟化方法。我们还讨论了虚拟机管理程序(hypervisor)的类型，然后讨论了 Xen 和 KVM 的高度概括的架构以及流行的开源的 Linux 虚拟化技术。

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. 在下一章中，我们将讨论 `libvirt`，`qemu` 和 KVM 的内部工作原理，并将获得这些组件是如何进行相互交流而实现虚拟化的相关知识。