KISS

SERVER VIRTUALIZATION

BY SHARATH KUMAR

Dedication

This book is dedicated to all those who seek to understand and harness the power of server virtualization. May it serve as a guide to help you navigate the complexities of this technology, and may it empower you to achieve your goals with ease and efficiency.

As Alan Watts once said, "The only way to make sense out of change is to plunge into it, move with it, and join the dance." So let us embrace the changes that server virtualization brings and dance with it to new heights.

Let this book be your partner in this journey, and let us explore the fascinating world of server virtualization together.

Preface

Server virtualization has revolutionized the way we use computing resources. The ability to run multiple virtual machines on a single physical server has enabled organizations to reduce hardware costs, improve efficiency, and increase flexibility. However, with so many components and tools involved, server virtualization can be a daunting topic to approach. That's why we've put together this comprehensive guide to server virtualization, with a focus on keeping things simple (KISS). From the definition and benefits of server virtualization to the components of a virtualization stack and steps to create and manage virtual machines, this guide covers all the essential aspects of server virtualization in a clear and concise manner. Whether you're new to server virtualization or looking to deepen your knowledge, this guide has something for you. So let's dive in and explore the world of server virtualization together

1. Introduction

In today's rapidly changing business environment, companies are facing many challenges that can make it difficult to stay competitive. For example, new technologies are constantly emerging, and businesses that fail to adapt risk falling behind. Additionally, global events such as pandemics or economic downturns can disrupt traditional ways of doing business and force companies to quickly adapt to new circumstances.

In this context, innovation and adaptability have become essential traits for businesses that want to succeed. It's not enough to simply rely on traditional methods or technologies - companies must be willing to embrace new approaches and technologies to stay ahead of the curve.

To help illustrate this point, let's take a look at a few stories that highlight the challenges that businesses can face in today's environment. These stories demonstrate how companies can struggle to adapt to new circumstances and technologies, and how innovation can be a key factor in staying competitive.

By examining these stories, we can better understand the importance of innovation and adaptability in modern business. We can also see how server virtualization can help businesses overcome some of these challenges and improve their operations.

Let's start with a story about a small marketing agency that found itself struggling to adapt to the new remote work environment brought on by the COVID-19 pandemic. Working from home was not a new concept, but the sudden and abrupt shift meant that the agency had to find a new way to power their website and store client data that would allow for remote access and flexibility.

Initially, the agency relied on physical servers housed in their office building to power their website and store client data. However, with everyone suddenly working from home, these servers became inaccessible, and the agency's employees were unable to access the necessary resources to do their work.

The marketing agency's IT team realized they needed to find a solution that would allow their employees to access the necessary resources from anywhere with an internet connection, while still ensuring that their data was secure. The IT team also had to make sure that the transition to virtualization did not disrupt the agency's workflow and did not compromise the quality of their services to clients.

After careful research and analysis, the IT team decided to move to server virtualization. This involved migrating their physical servers to a cloud-based virtual environment, which would allow their employees to access the necessary resources from anywhere with an internet connection. The virtualization process was complex and required meticulous planning, coordination, and implementation to ensure a smooth and successful transition.

Once the migration was complete, the benefits of virtualization were immediately apparent. The marketing agency's employees were able to access the necessary resources from anywhere with an internet connection, and they no longer had to be physically present in the office to access the servers. This increased their efficiency and productivity and allowed them to work flexibly from anywhere without worrying about accessing necessary resources.

Virtualization also allowed the agency to reduce their energy consumption and reduce their environmental impact by reducing their reliance on physical servers. The agency was able to scale their resources up and down as needed, further reducing costs.

The move to server virtualization was a game-changer for the marketing agency. It allowed them to adapt to the new remote work environment quickly and efficiently, while also reducing costs and minimizing their environmental impact. This story highlights how innovation can be a key factor in staying competitive in a rapidly changing world.

here's another story:

A small law firm had been operating for many years with a traditional on-premise IT infrastructure. Their servers were housed in a dedicated server room within their office, and their employees had desktop computers that were connected to the servers through a local network. However, as the firm grew and their client base expanded, they found that their IT infrastructure was becoming increasingly cumbersome and inefficient.

They had to frequently upgrade their hardware and software to keep up with their growing needs, which meant investing a lot of money and time into maintaining their servers and network. In addition, their employees were limited in terms of where they could work from. If an employee needed to work from home or while traveling, they were unable to access the firm's resources remotely.

The law firm's IT team realized that they needed to find a solution that would allow them to be more flexible and efficient, while also reducing their costs. After considering various options, they decided to move their IT infrastructure to a cloud-based virtual environment.

This involved migrating their servers to a cloud provider and giving their employees access to the resources they needed through a virtual private network (VPN). The migration process was complex and required careful planning, but once it was complete, the law firm enjoyed numerous benefits.

Their employees were now able to access the firm's resources from anywhere with an internet connection, which meant they could work from home or while traveling. This increased their productivity and allowed them to better serve their clients. In addition, the firm no longer had to invest in expensive hardware and software upgrades, which reduced their IT costs significantly.

Overall, the move to server virtualization allowed the law firm to modernize their IT infrastructure and become more flexible and efficient. It also allowed them to save money and focus on their core business, rather than worrying about maintaining their servers and network.

If you were an administrator working for the marketing agency or the law firm, you may have felt pressured to find a solution to the challenges brought on by the sudden shift to remote work. The responsibility of ensuring that the agency's website and client data were accessible to employees from anywhere with an internet connection while still maintaining data security could be overwhelming.

However, with the right tools and knowledge, you could approach the situation with confidence and overcome these challenges. This is where this book can be helpful. It can equip you with all the necessary tools, knowledge, and strategies to implement server virtualization successfully and ensure that the agency's operations run smoothly in a remote work environment.

The book can provide you with step-by-step guidance on the virtualization process, including how to plan and execute the virtual environment and how to manage and maintain the virtual servers effectively. With this information, you can approach the situation with confidence, knowing that you have the necessary tools to get the job done.

Overall, the book can help you navigate the challenges of server virtualization and ensure that the marketing agency can continue to serve its clients effectively in a rapidly changing world.

1.1. Definition

Server virtualization is a process of dividing a physical server into multiple virtual servers, each with its own operating system, applications, and resources. This technique allows multiple servers to run on a single physical machine, which helps reduce hardware costs and improves efficiency.

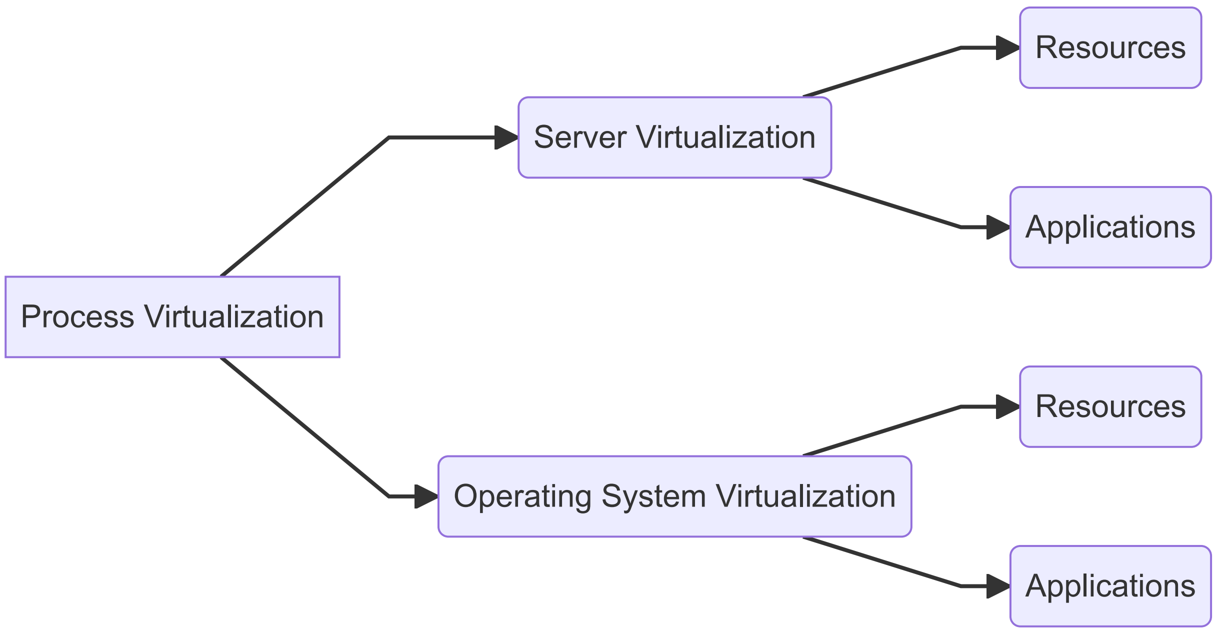
In the larger scheme of things, server virtualization is just one type of virtualization. Other types include access virtualization, application virtualization, process virtualization, storage virtualization, and network virtualization.

Access virtualization allows multiple users to access a single physical resource, such as a printer or scanner, as if they each had their own device. Application virtualization involves encapsulating an application and its dependencies into a virtual package, which can be run on any machine without requiring installation or configuration.

Storage virtualization involves combining multiple physical storage devices into a single virtual storage pool, which can be managed and allocated more efficiently. Network virtualization allows the creation of virtual networks, which can be customized and managed independently from the underlying physical network infrastructure.

Process virtualization, as mentioned earlier, is a type of virtualization that allows multiple operating systems or applications to run on a single physical machine at the same time. This technique is commonly used in server consolidation, where multiple servers are combined into a single physical machine to reduce hardware costs and improve efficiency.

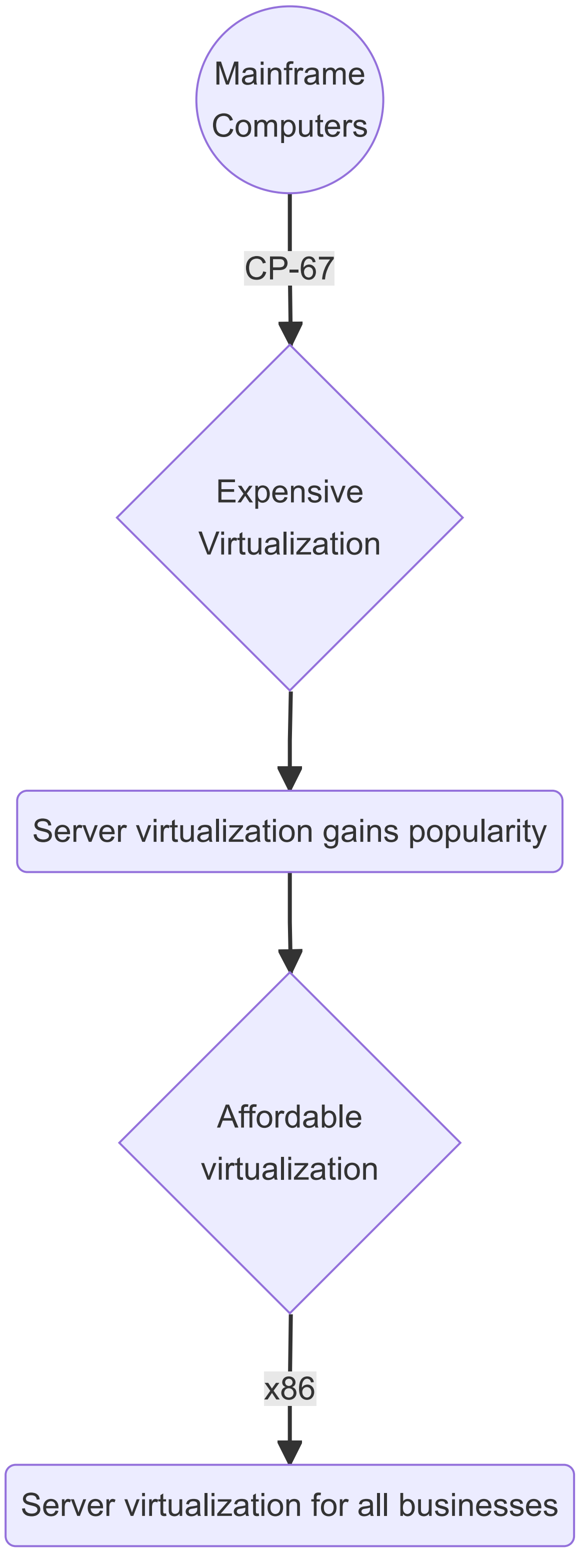
Server virtualization and operating system virtualization are both types of process virtualization. Server virtualization involves partitioning a physical server into multiple virtual servers, while operating system virtualization involves partitioning a single operating system into multiple virtual environments. Each approach has its own unique benefits and use cases, and the choice between them depends on the specific needs and requirements of the organization.


Process Virtualization
Server Virtualization
Resources
Applications
Operating System Virtualization
Resources
Applications

1.2. Benefits of Server Virtualization

- Improved Resource Utilization: Server virtualization allows for better utilization of physical server resources. With virtualization, multiple virtual machines can run on a single physical server, leading to more efficient use of server hardware and reduced costs associated with running and maintaining multiple physical servers.
- Reduced Number of Physical Servers: By consolidating multiple virtual machines onto a single physical server, server virtualization reduces the number of physical servers required in a data center. This leads to a reduction in physical space, power consumption, and cooling costs.
- Isolation of Virtual Machines: Server virtualization provides the ability to create virtual machines that are isolated from each other. This isolation can improve security and fault tolerance, as a problem with one virtual machine will not affect the others.
- Improved Fault Tolerance: With server virtualization, virtual machines can be set up with features such as automatic failover and load balancing. This can help ensure high availability and reduce the risk of downtime.
- Easy Server Management: Server virtualization allows for easier server management, as virtual machines can be easily created, cloned, and moved between physical servers. This allows for more flexibility and scalability, making it easier to manage large numbers of virtual machines.
- Cost Savings: By reducing the number of physical servers required, server virtualization can lead to significant cost savings in terms of hardware, power consumption, and maintenance costs.
- Green IT: By reducing the number of physical servers required and improving resource utilization, server virtualization can contribute to a more environmentally friendly IT infrastructure.

1.3. History of Server Virtualization


Mainframe Computers
CP-67
Expensive Virtualization
Server virtualization gains popularity
Affordable virtualization
x86
Server virtualization for all businesses

The concept of server virtualization has been around for several decades, with the first virtualization technology developed in the 1960s by IBM for use on its mainframe computers. In the early 2000s, server virtualization began to gain popularity as it became more affordable and accessible to smaller businesses.

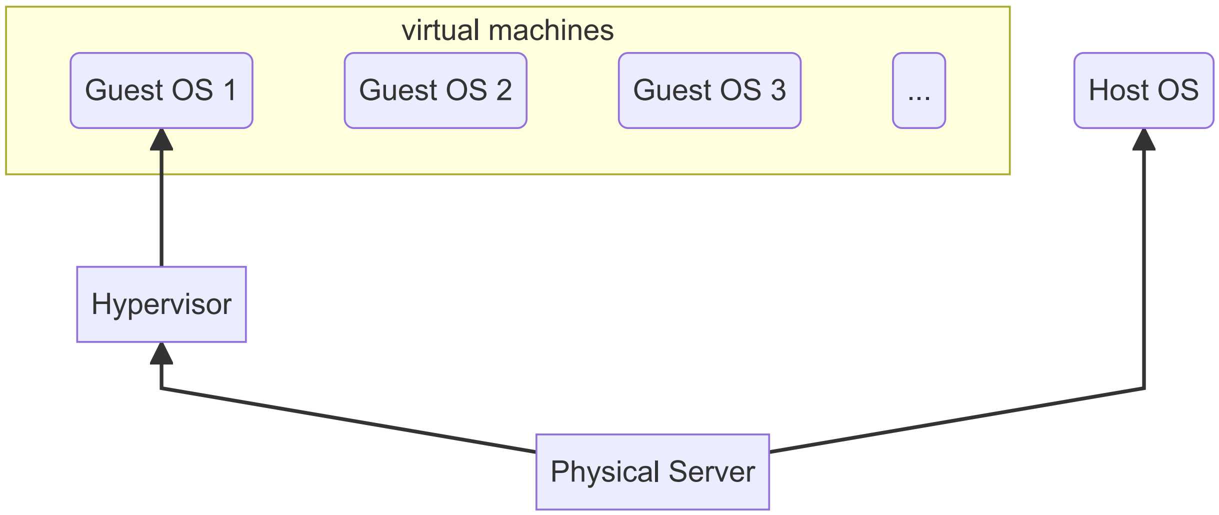
Server virtualization, as a concept, has been around for several decades. In fact, the first virtualization technology was developed by IBM in the 1960s for use on its mainframe computers. This technology, called CP-67, allowed multiple users to simultaneously access the same mainframe, with each user operating on their own virtual machine. *CP-67* is a hypervisor, or Virtual Machine Monitor, from IBM for its System/360 Model 67 computer.

However, at that time, virtualization was limited to large organizations with mainframe computers, and it was not until the early 2000s that server virtualization started to gain popularity among smaller businesses. This was due in part to the development of more affordable and accessible virtualization technologies, such as VMware's ESX Server.

With the introduction of these new technologies, businesses of all sizes were able to take advantage of server virtualization. By creating virtual machines that could run multiple operating systems and applications on a single physical server, businesses could save on hardware costs and improve the efficiency of their IT infrastructure.

Over time, virtualization technology continued to evolve and improve, with new features such as live migration and high availability becoming standard. Today, server virtualization is a critical component of modern IT infrastructure, allowing businesses to improve resource utilization, increase scalability, and enhance security and fault tolerance.

2. Server virtualization components



To achieve server virtualization, a software layer called a hypervisor is installed on the physical server. The hypervisor is responsible for creating and managing the virtual machines, as well as emulating the underlying physical components, such as the CPU, memory, and storage. Each virtual machine runs its own operating system, which can be different from the host operating system.

Server virtualization is a technique that allows multiple operating systems and applications to run on a single physical server simultaneously. It has become an essential component in modern computing infrastructure, enabling businesses to reduce their physical server requirements, improve utilization of server resources, and achieve cost savings.

To achieve server virtualization, there are three main components: the host, guest operating systems, and a hypervisor.

The hardware is the physical server itself - the computer that you can see and touch. This server has a certain amount of processing power, memory (RAM), and storage, all of which can be divided up and shared among multiple virtual machines.

The guest operating systems are the virtual machines that run on the host operating system. Each virtual machine has its own operating system, applications, and data. These virtual machines are isolated from each other, allowing multiple different operating systems to run on a single physical server simultaneously.

The hypervisor, also known as a virtual machine monitor, is the software layer that enables server virtualization. It sits between the host operating system and the guest operating systems, providing a virtualization layer that allows multiple operating systems to run on a single physical server simultaneously. The hypervisor is responsible for managing the virtual machines and their resource allocation.

2.1. Server Virtualization - Hardware

When it comes to server virtualization, having the right hardware is crucial. This is because virtualization requires a lot of resources, such as CPU, memory, and storage. Therefore, it's important to have hardware that's capable of handling the demands of virtualization.

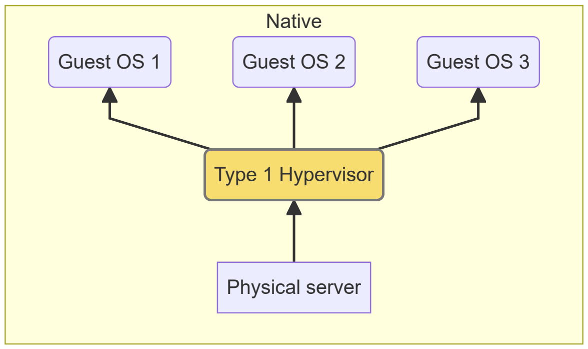
To support virtualization, modern CPUs from Intel and AMD come with hardware virtualization support. This feature is known as Intel VT-x (for Intel CPUs) and AMD-V (for AMD CPUs). Essentially, it allows the CPU to run multiple virtual machines simultaneously, by providing hardware-level support for virtualization.

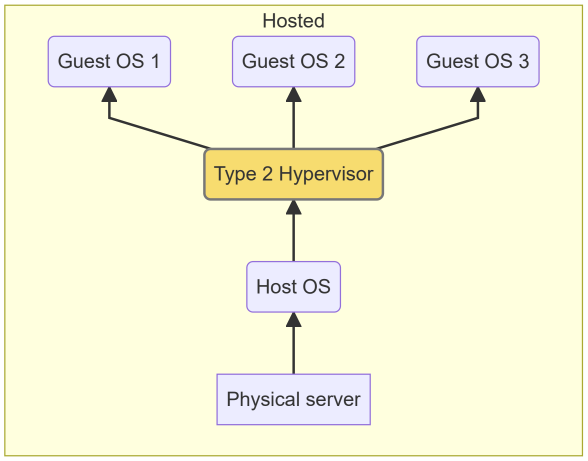
Another important component of server virtualization hardware is the BIOS (Basic Input/Output System). The BIOS is firmware that's built into the motherboard of a computer and is responsible for performing basic system tasks, such as booting up the computer and detecting hardware components.

In order for virtualization to work properly, the BIOS must have support for hardware virtualization. This can usually be enabled or disabled in the BIOS settings. Enabling hardware virtualization in the BIOS allows the CPU to access hardware-level support for virtualization, which can greatly improve performance and efficiency.

Overall, having the right hardware is essential for successful server virtualization. Hardware virtualization support from Intel and AMD, as well as BIOS support for hardware virtualization, are key components of a virtualization-ready system.

2.2. Server Virtualization: Hypervisor





A hypervisor is a type of software that enables server virtualization. It creates a layer of abstraction between the physical hardware and the virtual machines, allowing multiple operating systems to run on a single physical server.

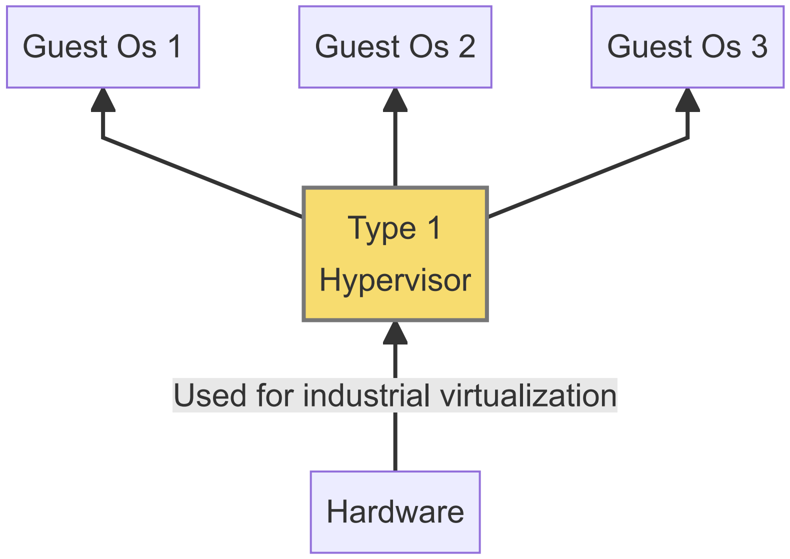
The term "hypervisor" comes from the word "supervisor", which refers to a manager who oversees a group of workers. In the case of server virtualization, the hypervisor acts as a supervisor for the virtual machines, managing their access to the physical resources of the server.

There are two types of hypervisors:

- Type 1 and Type 2. Type 1 hypervisors, also known as native or bare-metal hypervisors, run directly on the physical server hardware and provide direct access to the underlying physical resources. Examples of Type 1 hypervisors include VMware ESXi, Microsoft Hyper-V, and Citrix XenServer.
- Type 2 hypervisors, on the other hand, run on top of a host operating system and provide a virtualization layer between the guest operating systems and the host operating system. Examples of Type 2 hypervisors include Oracle VirtualBox, VMware Workstation, and Parallels Desktop.

Hypervisors, also known as virtual machine monitors, are the software or firmware components that enable virtualization on a physical server.

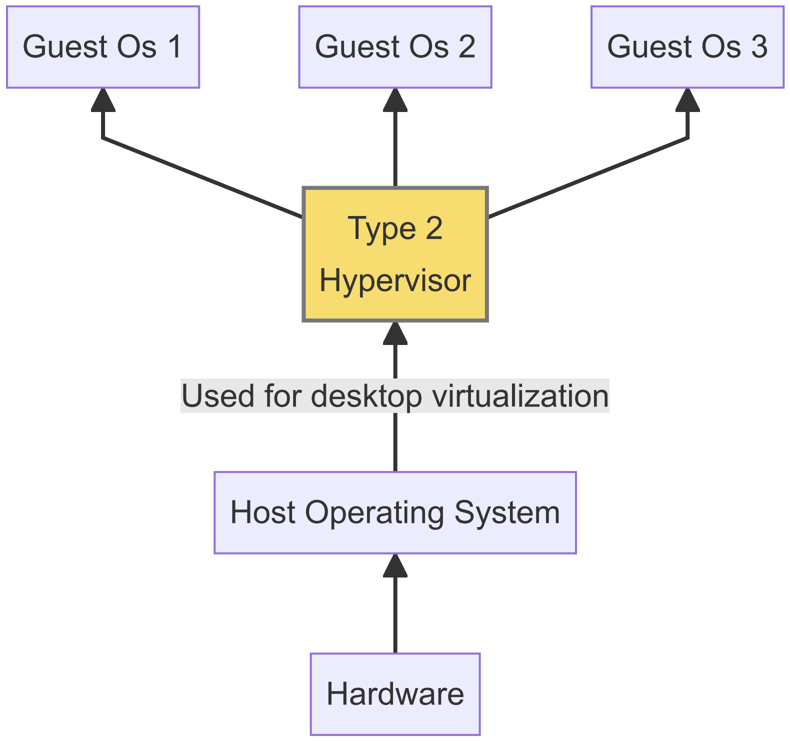
2.2.1. Type 1 Hypervisor



Also known as a native or bare-metal hypervisor, Type 1 hypervisors run directly on the host server's hardware and provide direct access to the physical resources of the server. These hypervisors have a small footprint and a minimal attack surface, which makes them ideal for enterprise deployments. Some examples of Type 1 hypervisors include:

1. VMware vSphere Hypervisor (ESXi): One of the most popular Type 1 hypervisors, ESXi is a bare-metal hypervisor that provides a robust virtualization platform for enterprise workloads. It supports a wide range of operating systems and provides advanced features such as vMotion, High Availability, and Distributed Resource Scheduler.
2. Microsoft Hyper-V: A Type 1 hypervisor that comes with Windows Server, Hyper-V provides a virtualization platform for running multiple operating systems on a single physical server. It supports both Windows and Linux guest operating systems and provides features such as live migration, high availability, and storage migration.
3. Citrix Hypervisor: Formerly known as XenServer, Citrix Hypervisor is an open-source Type 1 hypervisor that provides enterprise-class virtualization capabilities. It supports multiple guest operating systems and provides features such as live migration, high availability, and disaster recovery.

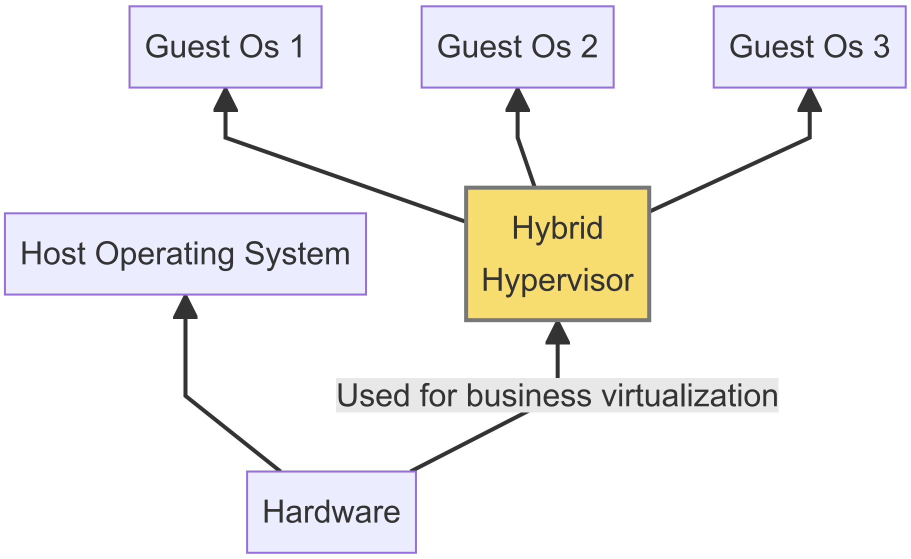
2.2.2. Type 2 Hypervisor



Also known as a hosted hypervisor, Type 2 hypervisors run on top of a host operating system and are more commonly used for desktop virtualization. These hypervisors are easier to install and manage but may have performance limitations due to their reliance on the underlying operating system. Some examples of Type 2 hypervisors include:

1. Oracle VirtualBox: A popular Type 2 hypervisor, VirtualBox is a free and open-source virtualization platform that allows users to run multiple guest operating systems on a single host operating system. It provides features such as snapshotting, cloning, and remote display.
2. VMware Workstation: A Type 2 hypervisor designed for desktop use, Workstation allows users to create and run multiple virtual machines on a single host operating system. It provides features such as snapshots, cloning, and remote access.
3. Parallels Desktop: A Type 2 hypervisor designed for macOS, Parallels Desktop allows users to run multiple operating systems on their Mac computer. It provides features such as Coherence mode, which allows users to run Windows applications on their Mac desktop.

2.2.3. Hybrid Hypervisor



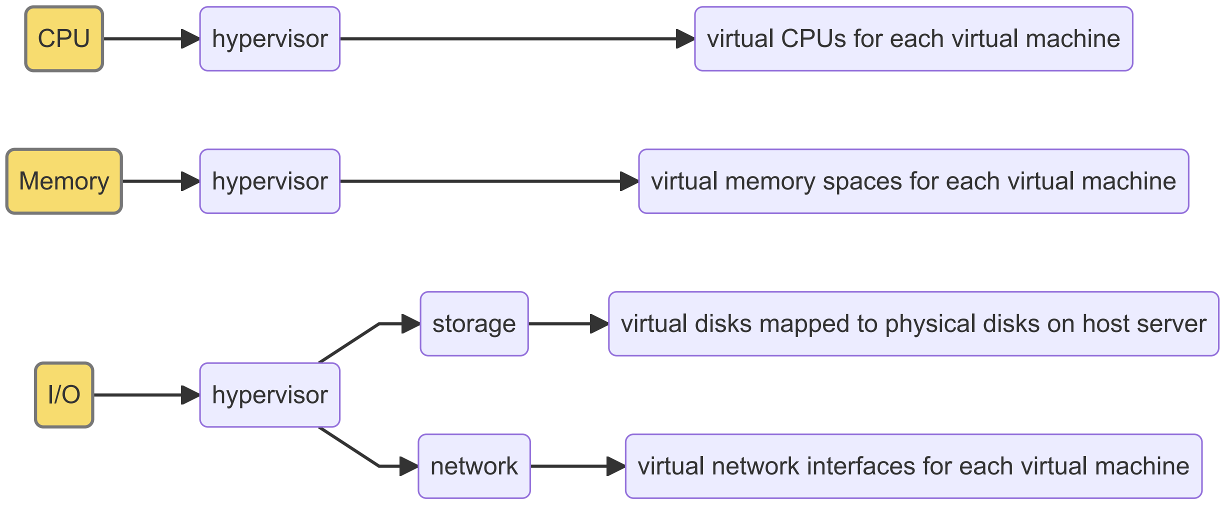
Hybrid hypervisors, also known as nested hypervisors, combine elements of both type 1 and type 2 hypervisors. They allow for virtualization within virtualization by running a type 2 hypervisor on top of a type 1 hypervisor. One example of a hybrid hypervisor is Linux KVM with QEMU.

Here is a table summarizing the differences between the three types of hypervisors:

	Type 1 Hypervisor	**Type 2 Hypervisor**	**Hybrid Hypervisor**
Runs on	Host machine's hardware	Host operating system	Host machine's hardware
Access to hardware	Direct access	Through host OS	Direct access
Performance	High	Decreased	Varies
Use cases	Enterprise, high performance	Desktop virtualization, testing	

	Type 1 Hypervisor	Type 2 Hypervisor	Hybrid Hypervisor
Examples	VMware ESXi, Microsoft Hyper-V, Citrix XenServer	Oracle VirtualBox, VMware Workstation, Parallels Desktop	Linux KVM with QEMU

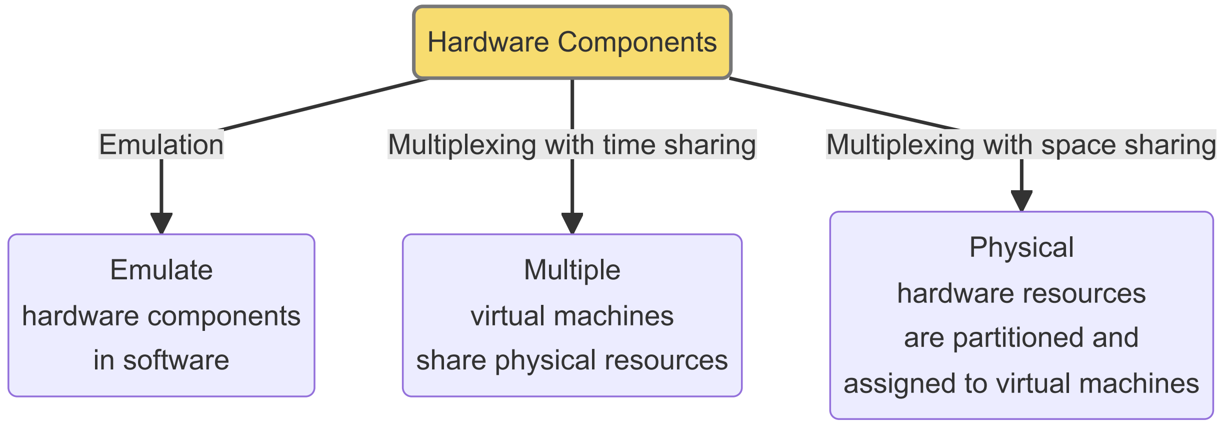
2.3. Server Virtualization: VMs


CPU
hypervisor
virtual CPUs for each virtual machine
Memory
hypervisor
virtual memory spaces for each virtual machine
I/O
hypervisor
storage
virtual disks mapped to physical disks on host server
network
virtual network interfaces for each virtual machine

The virtualization of hardware components is crucial to ensure that each virtual machine can have its own CPU, memory, and I/O resources, which are isolated from other virtual machines running on the same physical server. Here are the hardware components that need to be virtualized in server virtualization:

1. CPU: The CPU is virtualized by the hypervisor, which allocates a portion of the CPU's time(time sharing) or a portion of the CPU(LPAR -logical partitioning) to each virtual machine. The hypervisor creates virtual CPUs (vCPUs) for each virtual machine, and each vCPU is scheduled to run on a physical CPU core.
2. Memory: The memory is virtualized by the hypervisor, which creates virtual memory spaces for each virtual machine. Each virtual machine thinks it has access to a certain amount of physical memory, but the hypervisor actually manages the physical memory and maps it to the virtual memory spaces.
3. I/O: I/O is virtualized in two ways: storage and network. For storage, the hypervisor can provide virtual disks to each virtual machine, which are mapped to physical disks on the host server. For network, the hypervisor creates virtual network interfaces for each virtual machine, which can be connected to virtual switches that are connected to physical network interfaces on the host server.

2.4. Ways to Virtualize Components


Hardware Components
Emulation
Multiplexing with time sharing
Multiplexing with space sharing
Emulate hardware components in software
Multiple virtual machines share physical resources
Physical hardware resources are partitioned and assigned to virtual machines

There are several ways to virtualize these hardware components:

1. Emulation: In emulation, the hypervisor emulates the hardware components of the physical server in software. This is useful when the guest operating system running in the virtual machine is different from the host operating system. For example, when running a Windows x86 virtual machine on a mac host, the hypervisor may need to emulate the x86 components of a Windows system.
2. Multiplexing with time sharing: In this method, the physical hardware resources are shared among multiple virtual machines. The hypervisor schedules the use of CPU time, memory, and I/O resources among the virtual machines, so that each virtual machine gets a fair share of the resources.
3. Multiplexing with space sharing: In this method, the physical hardware resources are divided into partitions, and each partition is assigned to a virtual machine. Each virtual machine has its own dedicated CPU, memory, and I/O resources, which are not shared with other virtual machines.

Virtualization is the creation of virtual versions of hardware components. On the other hand, emulation involves the replication of one system on another, often with different hardware, software, or both.

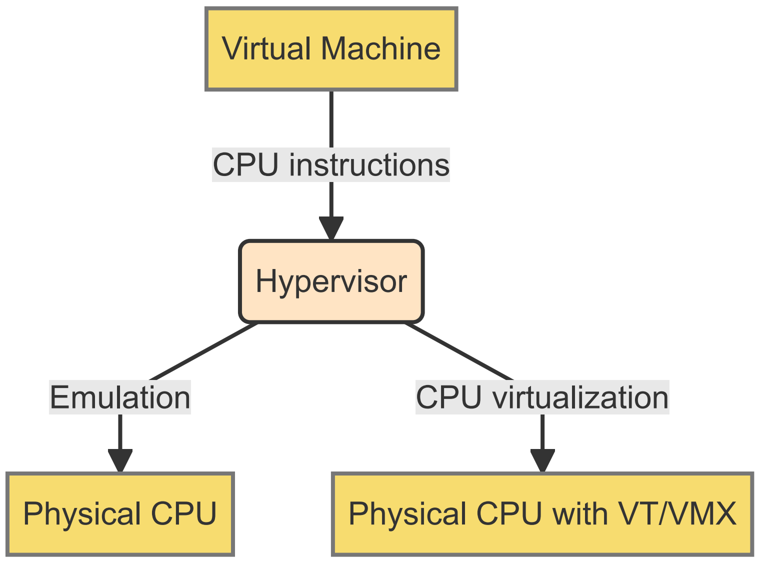
Here's a table summarizing the differences between virtualization and emulation:

	Virtualization	**Emulation**

	Virtualization	**Emulation**
Purpose	Run multiple operating systems on one machine	Run one operating system on another
Performance	Efficient, close to native performance	Inefficient, slower than native performance
Hardware	Uses the same hardware as the host machine	May require different hardware than the host machine
Software	Runs on the host machine's operating system	Requires an emulator to run on the host machine

Now that we have a better understanding of virtualization versus emulation, let's delve into CPU virtualization.

2.4.1. CPU virtualization



CPU emulation is a technique used in virtualization to simulate the behavior of a physical CPU within a virtual machine. Emulating a CPU involves creating a software layer that translates the instructions and operations of the virtual machine into instructions that can be executed by the physical CPU. This technique is commonly used in virtual machines that run on a different CPU architecture than the physical host machine.

One of the main downsides of CPU emulation is that it can be slow and resource-intensive. Emulating a CPU involves translating every instruction and operation, which can result in a significant performance overhead. This can be especially problematic in virtual environments where multiple virtual machines are running on the same physical host machine.

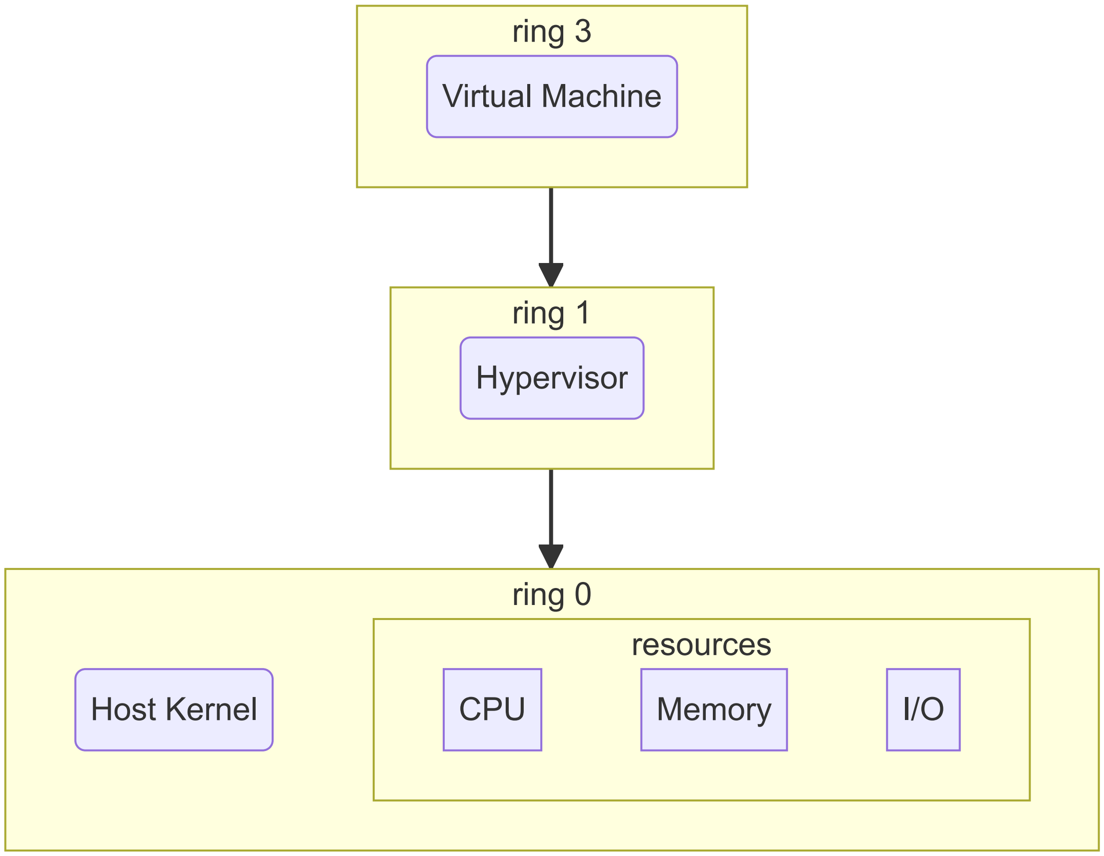
CPU virtualization is a special case of CPU emulation that attempts to mitigate the performance overhead by using hardware support for virtualization. CPU virtualization involves creating a virtual machine that can directly execute instructions on the physical CPU without the need for translation.

There are two main ways to achieve CPU virtualization: software-based and hardware-based. Software-based CPU virtualization involves using a hypervisor or virtual machine monitor to manage access to the physical CPU. The hypervisor intercepts instructions from the virtual machine and translates them into instructions that can be executed by the physical CPU.

Hardware-based CPU virtualization, on the other hand, relies on hardware support for virtualization. This involves using a special CPU feature known as Virtualization Technology (VT) or Virtualization Extensions (VMX) that allows the hypervisor to create and manage virtual machines that can access the physical CPU directly.

Overall, CPU virtualization provides a more efficient way to virtualize the CPU than emulation, and can help to improve performance and reduce overhead in virtual environments.

2.4.1.1. CPU virtualization Challenge : Protection Rings



2.4.1.1.1. Protection Rings

So, have you ever heard of protection rings before? They're actually a really important concept when it comes to virtualization security. Protection rings are a way to define levels of privilege within a computer system. In particular, they're used to help isolate virtual machines in a hypervisor.

There are four protection ring levels, with level 0 being the most privileged and level 3 being the least privileged. Level 0 is known as kernel mode, and it's where the operating system kernel resides. This is where the most sensitive and critical operations of the system take place. Levels 1 and 2 are typically reserved for device drivers and other system software, while level 3 is user mode, where most applications run.

2.4.1.1.2. System Calls

System calls are an essential part of any operating system. They are the way that programs running on a computer can interact with the operating system and request services like reading or writing to a file, accessing a network resource, or allocating memory. When a program makes a system call, it triggers an interrupt that switches control from the program to the operating system. The operating system then performs the requested service and returns control back to the program.

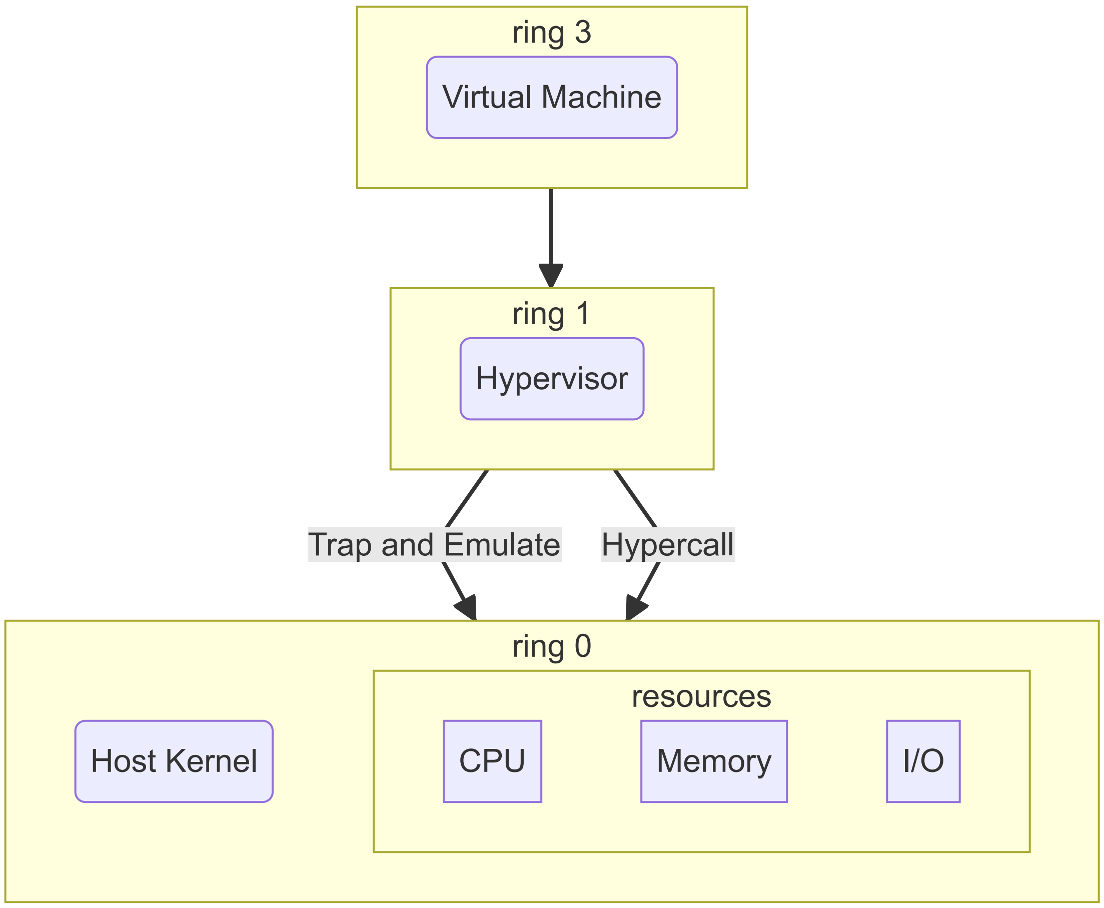
In the context of virtualization, system calls take on a crucial role in allowing virtual machines to access the resources of the physical machine on which they are running. When a virtual machine running on a hypervisor needs to access a resource like a network interface or a disk drive, it makes a system call to the hypervisor, which then handles the request and maps it to the underlying physical hardware.

The way that system calls are implemented in virtualization depends on the type of hypervisor being used. In a Type 1 hypervisor, also known as a "bare metal" hypervisor, the hypervisor runs directly on the physical hardware of the machine, and virtual machines run on top of the hypervisor. When a virtual machine makes a system call, the hypervisor intercepts it and handles it directly, without needing to go through the host operating system. In contrast, in a Type 2 hypervisor, the hypervisor runs as an application on top of the host operating system, and virtual machines run on top of the hypervisor. In this case, when a virtual machine makes a system call, the hypervisor intercepts it and translates it into a call to the host operating system, which then handles the request.

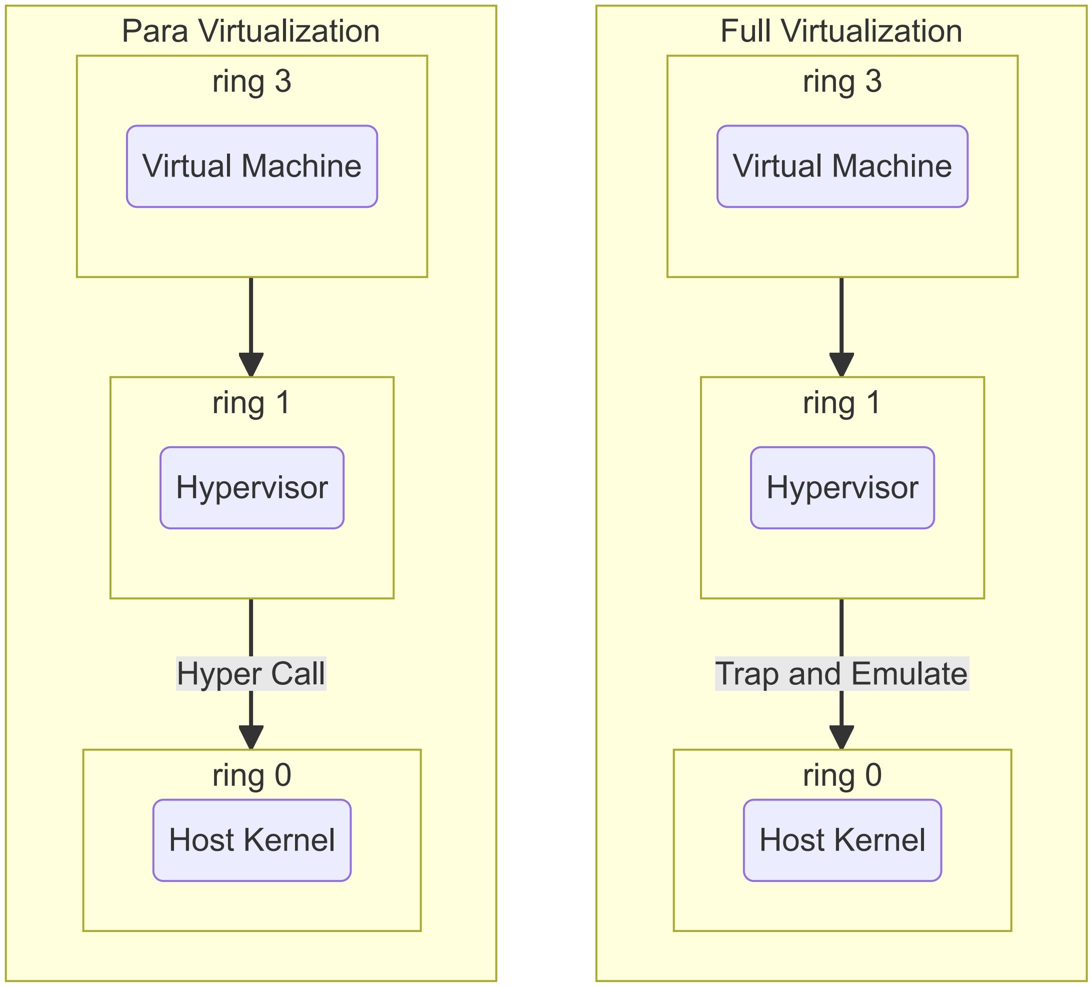
At the heart of server virtualization lies the challenge of executing kernel instructions (*system calls*) of virtual machines (VMs) due to the *protection rings*. In particular, kernel instructions (system calls) execute only with ring 0, which is reserved for the kernel itself, and we cannot have the VM kernel and the host kernel running on the same ring 0.

This challenge is particularly significant because the kernel is the core of the operating system that manages hardware resources and provides services to applications. As such, executing kernel instructions of the VMs is essential for achieving full process virtualization. However, doing so without disrupting the host kernel's functionality requires a clever solution.

2.4.1.2. CPU virtualization Solution



To overcome this challenge, the solution of full virtualization was proposed. In this approach, the host kernel runs in ring 0, and the VM runs in ring 1 or 3. Any VM kernel instructions that get executed in ring 1 will raise a trap that can be caught by the VMM/hypervisor and then emulated. This strategy is called Trap and Emulate.



The concept of full virtualization provides a solution to the challenge of executing kernel instructions of the VMs due to the protection rings. Full virtualization is a technique that allows the VM to run its own operating system kernel, providing the illusion that it is running on its own physical hardware.

In this approach, the host kernel runs in ring 0, and the VM runs in ring 1 or 3. Ring 0 is the most privileged protection ring, where the host kernel runs, and ring 1 and 3 are less privileged protection rings, where the VM kernel runs. By running the VM in a less privileged protection ring, the VMM/hypervisor can trap any VM kernel instructions that execute in ring 1 or 3, and then emulate those instructions in ring 0.

The trap and emulate strategy is the key component of the full virtualization technique. When a VM attempts to execute a privileged instruction in ring 1 or 3, it causes a trap or exception to occur, which the VMM/hypervisor intercepts. The VMM/hypervisor then emulates the instruction in ring 0 and returns the result to the VM, providing the illusion that the instruction executed natively on the hardware. This trap and emulate strategy is used to virtualize CPU instructions and is the core technique used in most hypervisors to provide full virtualization.

To overcome the performance overhead caused by the trap and emulate strategy, the solution of paravirtualization was proposed. In this approach, the guest operating system is modified to be aware that it is running in a virtualized environment. This allows the guest OS to communicate directly with the hypervisor and avoid the trap and emulate overhead.

Paravirtualization provides a solution to the challenge of executing kernel instructions of the VMs due to the protection rings. It is a technique that allows the VM to run its own operating system kernel, providing the illusion that it is running on its own physical hardware. However, the guest OS must be modified to be aware that it is running in a virtualized environment.

In contrast to full virtualization, where the VM runs in a less privileged protection ring, paravirtualization requires the VM to run in a separate address space. The hypervisor provides an interface to the VM that allows it to communicate directly with the physical hardware, bypassing the trap and emulate overhead. This approach can provide better performance than full virtualization because it eliminates the need for the VMM/hypervisor to emulate privileged instructions.

Both full virtualization and paravirtualization are used to virtualize CPU instructions and are the core techniques used in most hypervisors to provide virtualization. The choice between full virtualization and paravirtualization depends on the specific use case and the performance requirements of the virtualized environment.

In summary, the trap and emulate strategy is the key component of full virtualization, while paravirtualization allows the guest OS to communicate directly with the hypervisor and avoid the trap and emulate overhead. Both approaches are used to virtualize CPU instructions, and the choice depends on the specific use case and performance requirements.

	Full Virtualization	**Paravirtualization**
VM runs in	Ring 1 or 3	Separate address space
Guest OS modification required?	No	Yes

	Full Virtualization	**Paravirtualization**
Overhead	Trap and emulate	Direct communication with hypervisor
Performance	Lower	Higher

2.4.1.3. x86 CPU Virtualization Challenge:

The x86 architecture poses a significant challenge for virtualization because it does not fully adhere to the Popek and Goldberg principle, which states that the sensitive instructions of the CPU architecture must be a subset of the privileged architecture.

Popek and Goldberg is a principle in computer science that determines whether a given computer architecture can be virtualized or not. The principle was proposed by Gerald J. Popek and Robert P. Goldberg in their 1974 paper titled "Formal Requirements for Virtualizable Third Generation Architectures."

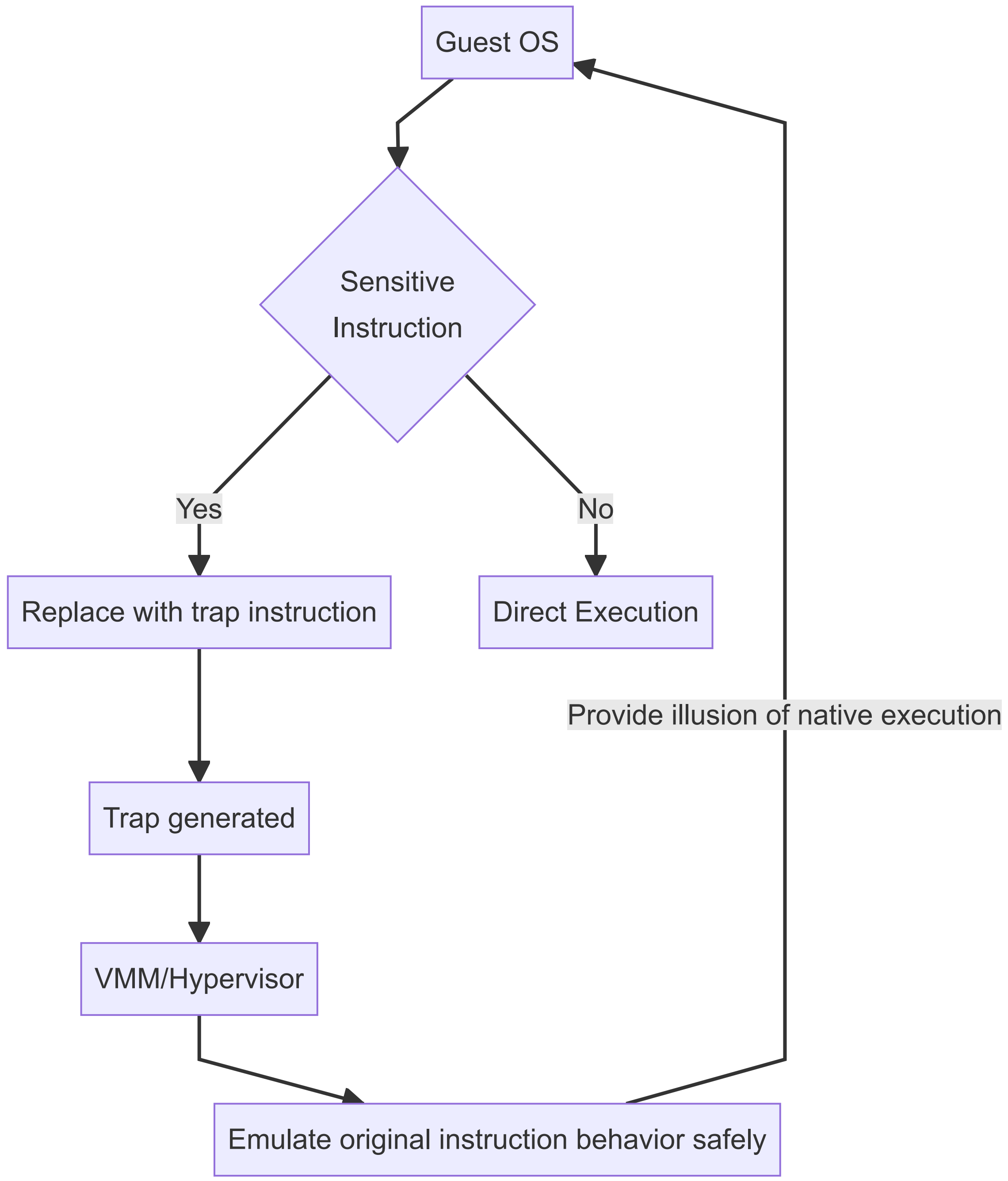
According to the Popek and Goldberg principle, an architecture is virtualizable if the sensitive instructions of the architecture can be a subset of the privileged instructions. In other words, if the architecture allows the virtual machine to trap and emulate any sensitive instructions that are executed by the guest operating system, then the architecture can be virtualized.

The Popek and Goldberg principle defines three categories of instructions: privileged, sensitive, and non-sensitive. Privileged instructions are the ones that can only be executed by the operating system kernel, and they have direct access to the hardware. Sensitive instructions are the ones that change the behavior of the hardware, and they require a trap and emulation mechanism to be executed safely in a virtual environment. Non-sensitive instructions, on the other hand, do not affect the behavior of the hardware and can be executed directly by the virtual machine.

The Popek and Goldberg principle is an essential guideline for developing virtualization technologies, and it has been used to determine the virtualizability of various computer architectures, such as IBM System/370, DEC VAX, and Intel x86. While some architectures, like IBM System/370, follow the principle, others, like Intel x86, do not fully comply with it, making virtualization a challenging task.

2.4.1.4. x86 CPU Virtualization Solution:

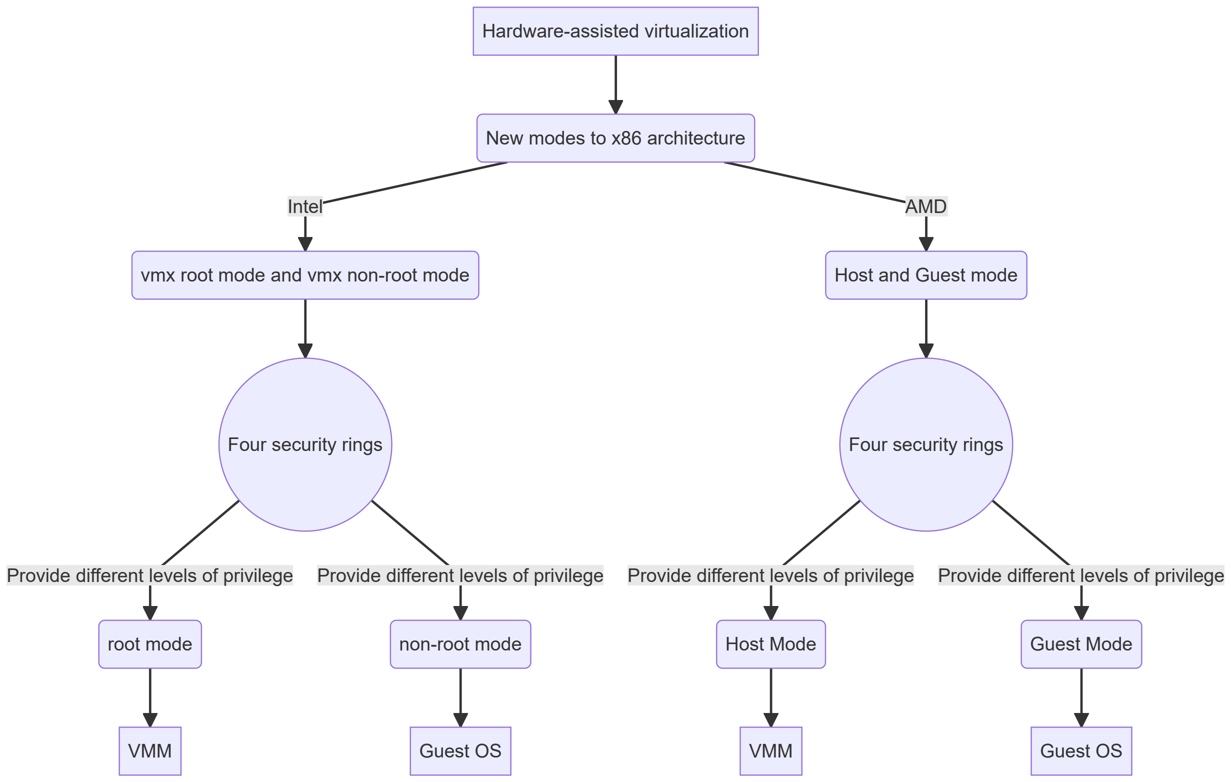
2.4.1.4.0.1. Software solution


Guest OS
Sensitive Instruction
Yes
No
Replace with trap instruction
Direct Execution
Provide illusion of native execution
Trap generated
VMM/Hypervisor
Emulate original instruction behavior safely

VMware introduced binary translation as a software solution to overcome the challenge of virtualizing the x86 architecture. This method works by analyzing the code of the guest operating system and replacing any sensitive instructions that could cause failure in the virtual environment with instructions that generate a trap. When the VM tries to execute the replaced instructions, a trap is generated, and control is passed to the VMM/hypervisor. The VMM/hypervisor then emulates the original instruction's behavior in a safe manner and returns the result to the VM, providing the illusion that the instruction was executed natively on the hardware.

Binary translation allows for the full virtualization of the x86 architecture, and it is widely used in modern hypervisors. The downside of this approach is that the translation process can be time-consuming and can impact the overall performance of the virtual environment. However, this overhead can be reduced by optimizing the translation process and caching frequently used translations. Overall, binary translation is an effective software solution for virtualizing x86 architecture and has enabled the widespread adoption of virtualization in modern computing.

2.4.1.4.0.2. Hardware solution


Hardware-assisted virtualization
New modes to x86 architecture
Intel
AMD
vmx root mode and vmx non-root mode
Host and Guest mode
Four security rings
Four security rings
Provide different levels of privilege
Provide different levels of privilege
Provide different levels of privilege
Provide different levels of privilege
root mode
non-root mode
Host Mode
Guest Mode
VMM
Guest OS
VMM
Guest OS

Hardware-assisted virtualization is a technique that uses hardware to provide support for virtualization. It allows the VMM/hypervisor to create and manage virtual machines with minimal performance overhead. One of the main issues with x86 architecture virtualization is that it does not conform to the Popek and Goldberg principle, which makes it challenging to virtualize.

To address this limitation, hardware-assisted virtualization introduced new security rings to the x86 architecture, such as the -1 ring in the x86 architecture and the vmx root mode and vmx non-root mode in Intel architecture. In vmx root mode, the VMM/hypervisor is executed, and in vmx non-root mode, the guest OS runs. Both modes have four security rings, and these rings provide different levels of privilege to the VMM/hypervisor and the guest OS. By providing different levels of privilege, the VMM/hypervisor can handle sensitive instructions, reducing the need for binary translation, which can cause performance overhead.

Like Intel's vmx mode, AMD's Secure Virtual Machine (SVM) also uses a hardware-based approach to virtualize the x86 architecture. SVM provides hardware support for virtualization that is compatible with existing x86 architectures.

AMD's SVM has a similar approach to Intel's vmx mode, but it uses different terminology. Instead of vmx root mode and vmx non-root mode, AMD's SVM has Host Mode and Guest Mode. In Host Mode, the VMM/hypervisor is executed, and in Guest Mode, the guest OS runs. The SVM also has four security rings that provide different levels of privilege to the VMM/hypervisor and the guest OS.

In AMD's SVM, the VMM/hypervisor runs in Host Mode, which is equivalent to Intel's vmx root mode. Host Mode provides full access to the hardware resources and allows the VMM/hypervisor to execute privileged instructions. The guest OS runs in Guest Mode, which is equivalent to Intel's vmx non-root mode. In Guest Mode, the guest OS has limited access to hardware resources and cannot execute privileged instructions.

Overall, AMD's SVM and Intel's vmx mode have similar hardware-assisted virtualization approaches for the x86 architecture, but with different terminology. Both modes use security rings to provide different levels of privilege to the VMM/hypervisor and the guest OS, allowing for better performance and fewer limitations compared to software-based solutions.

3. Virtualization Stack

QEMU was first released in 2003 and was originally designed to emulate various systems such as PowerPC, x86, and ARM. Later on, QEMU added support for virtualization, allowing it to run virtual machines with near-native performance. QEMU can emulate a wide range of system architectures, from PowerPC and ARM to x86 and MIPS.

One of the key advantages of QEMU is its ability to run virtual machines across a variety of hardware architectures, including both 32-bit and 64-bit systems. It can also run on a wide range of host operating systems, including Linux, macOS, and Windows. Additionally, QEMU can be used as a standalone tool or in combination with other virtualization technologies such as KVM.

There are different types of virtualization, but today I want to talk specifically about using the QEMU hypervisor.

So, when we talk about virtualization using QEMU it's important to understand the different layers of the virtualization stack. At the base layer, we have KVM which is a type 1 hypervisor that runs directly on top of the host machine's hardware. It provides hardware-level virtualization capabilities, which means that it enables multiple operating systems to run on a single physical machine without interfering with each other.

On top of KVM, we have QEMU which is a type 2 hypervisor that runs in user space. While QEMU on its own is only a type 2 hypervisor, when you add KVM to it, it converts it into a type 1 hypervisor. QEMU is responsible for emulating virtual hardware and running guest operating systems. It also provides features like disk and network emulation, which enables guests to communicate with the outside world. Now, creating virtual machines using QEMU directly can be a bit of a hassle. It requires a lot of command line options and configuration files, which can be tedious and error-prone. That's where the libvirt software comes in. It simplifies the process of creating virtual machines by exposing a user-friendly API, so you don't have to worry about all those command line options.

Now, on top of QEMU, we have Libvirt. Libvirt is an open-source virtualization API that abstracts away the underlying virtualization technologies, including KVM, QEMU, and others. This means that Libvirt can communicate with different hypervisors, and provide a common API to manage them all. Libvirt provides a high-level API that can be used to create, configure, and manage virtual machines. The cool thing about libvirt is that it works with multiple hypervisors, including QEMU, Hyper-V, and others. So, you only need to learn one API to work with all these different virtualization technologies.

Finally, at the top of the stack, there are tools like virsh and VMM since dealing with the libvirt API can still be a bit tricky. Virsh is a command-line tool for managing virtual machines using libvirt, while VMM (Virtual Machine Manager) is a graphical interface for managing virtual machines. Another tool called Cockpit provides a web-based interface for managing virtual machines, making it accessible from anywhere with an internet connection. These tools interact with libvirt under the hood, so you don't have to worry about the nitty-gritty details.

Let me explain the virtualization stack consisting of QEMU, KVM, libvirt, and the virt tools like virsh and vmm using a car analogy.

Think of a Corvette car as your virtual machine. The Corvette has its own engine, wheels, and other components that allow it to function independently. Similarly, a virtual machine has its own operating system, software, and other resources that allow it to function independently.

Now, imagine that you want to control the Corvette car from a distance. You could use a remote control, but you need a way to communicate with the car. This is where the QEMU hypervisor comes in. It acts as a bridge between the virtual machine and the physical machine, allowing you to control the virtual machine from your host machine.

But there's a problem: the QEMU hypervisor alone can only operate in Type 2 mode, which is like using the remote control from a distance. To get closer to the Corvette and have more control, you need to switch to Type 1 mode. This is where KVM comes in. KVM is a Type 1 hypervisor that runs directly on the host machine's hardware, allowing for faster and more efficient control of the virtual machine.

Now imagine you have a whole garage full of cars, each with different engines, wheels, and other components. It would be a nightmare to try to control each car individually. This is where libvirt comes in. Libvirt acts as an abstraction layer, allowing you to manage multiple virtual machines from a single interface, regardless of which hypervisor they're running on.

Finally, virt tools like virsh and vmm are like the dashboard of your car. They allow you to monitor and manage your virtual machines through libvirt, just like the dashboard of your car allows you to monitor and manage the car's various systems.

To give an example, let's say you're a software developer working on a web application. You need to test your application on different operating systems, but you don't want to buy multiple physical machines. Instead, you can use QEMU and KVM to create virtual machines running different operating systems. You can then use libvirt to manage these virtual machines and virt tools like virsh and vmm to monitor and manage them. This allows you to test your application on multiple operating systems without the need for multiple physical machines.

Here's a table of tools:

Virtualization Tool	**Advantages**	**Disadvantages**
VMM	Easy-to-use graphical interface	Limited customization options
Cockpit	Web-based interface accessible from anywhere	Limited customization options
`virsh`, `virt-install`	Command line interface which is lightweight and fast	Extensive customization options

3.1. Virtualization with VMM and Virsh

First, let me give you a brief explanation of what VMM and Virsh are. VMM stands for Virtual Machine Manager, and it's a tool that allows you to create and manage virtual machines. Virsh, on the other hand, is a command-line tool that lets you manage virtual machines on a host system.

The advantage of using these tools is that they provide a level of abstraction between the virtual machine and the underlying hardware. This allows you to run multiple virtual machines on a single physical machine, each with their own operating system and resources like CPU, memory, and storage.

Using virtualization with VMM and Virsh can be incredibly useful, especially in situations where you need to test software or run multiple applications that require different operating systems or configurations. For example, if you're a web developer, you might need to test your application on multiple operating systems and browser configurations. Instead of buying multiple physical machines, you could use virtualization to run all of these configurations on a single physical machine.

Another advantage of virtualization is that it allows you to more easily manage and migrate virtual machines between physical hosts. This is especially useful in cloud computing environments where virtual machines can be easily moved between physical hosts to optimize resource usage.

Overall, virtualization with VMM and Virsh can be a powerful tool that can save you time, money, and resources by allowing you to run multiple virtual machines on a single physical machine, and to easily manage and migrate virtual machines between physical hosts.

3.2. QEMU

QEMU, short for Quick Emulator, is a powerful and versatile software emulator that allows you to run virtual machines on your physical machine. Essentially, QEMU creates a virtual environment that emulates a computer's hardware, allowing you to run operating systems and applications that are designed to run on different types of hardware.

QEMU is a type 2 hypervisor, which means that it runs on top of a host operating system. This is different from a type 1 hypervisor like KVM, which runs directly on the host's hardware. However, QEMU can work in conjunction with KVM to provide a complete virtualization solution.

One of the benefits of using QEMU is that it can emulate a wide range of hardware components, including CPUs, memory, storage devices, and network interfaces. This means that you can use QEMU to create virtual machines that are running a different operating system or have different hardware requirements than your physical machine.

Another benefit of using QEMU is that it supports a variety of virtual machine formats, including raw, qcow, and vmdk. This means that you can easily create and manage virtual machines using QEMU in a format that is compatible with other virtualization platforms.

So, in summary, QEMU is a powerful and versatile software emulator that allows you to run virtual machines on your physical machine. It's a type 2 hypervisor that can work in conjunction with KVM to provide a complete virtualization solution. It supports a wide range of hardware components and virtual machine formats, making it a flexible and useful tool for virtualization.

3.3. KVM (Kernel-based Virtual Machine)

KVM, short for Kernel-based Virtual Machine, is a virtualization technology that is built into the Linux kernel. It allows you to create and run virtual machines on your Linux-based system.

KVM is a type 1 hypervisor, which means that it runs directly on the host's hardware. This is different from a type 2 hypervisor like QEMU, which runs on top of a host operating system. KVM is designed to provide efficient and high-performance virtualization, making it a popular choice for running virtual machines on Linux-based servers and workstations.

One of the key benefits of KVM is that it works in conjunction with QEMU to provide a complete virtualization solution. QEMU provides the emulation layer that allows you to run virtual machines, while KVM provides the hypervisor layer that allows you to run virtual machines with near-native performance.

So, how does KVM convert QEMU into a type 1 hypervisor? When you use KVM, QEMU is integrated into the Linux kernel as a kernel module. This allows QEMU to take advantage of the hardware virtualization capabilities that are built into modern CPUs, such as Intel VT-x and AMD-V. These hardware virtualization features allow KVM to create virtual machines that run with near-native performance, making it a highly efficient virtualization solution.

In summary, KVM is a virtualization technology that is built into the Linux kernel. It's a type 1 hypervisor that provides efficient and high-performance virtualization. KVM works in conjunction with QEMU to provide a complete virtualization solution, with QEMU providing the emulation layer and KVM providing the hypervisor layer. Together, they create a powerful and flexible virtualization platform that is widely used in the industry.

3.4. Libvirt

Libvirt is an open-source virtualization management framework that provides a common interface for managing various virtualization technologies, including KVM, QEMU, Xen, VirtualBox, and more. It's designed to make it easier to manage virtualization technologies, no matter what underlying hypervisor you're using.

So, how does libvirt work? At its core, libvirt provides an API that allows you to manage virtual machines, storage, and networks through a unified interface. This API can be accessed using a variety of programming languages, including C, Python, and Java.

When you use libvirt, it abstracts the underlying virtualization technologies and provides a common interface that you can use to manage them. For example, if you want to create a new virtual machine, you can use libvirt's API to do so, and libvirt will take care of communicating with the underlying hypervisor to create the virtual machine. Similarly, if you want to attach a new network interface to a virtual machine, you can use libvirt's API to do so, and libvirt will take care of the underlying details.

One of the biggest benefits of using libvirt is that it works with many different hypervisors. This means that you can use the same tools and techniques to manage virtual machines, storage, and networks, no matter which underlying hypervisor you're using. This can simplify management and make it easier to switch between different virtualization technologies, as you won't need to learn new tools and techniques every time you switch.

In summary, libvirt is an open-source virtualization management framework that provides a common interface for managing various virtualization technologies. It works by abstracting the underlying hypervisors and providing a unified API that you can use to manage virtual machines, storage, and networks. One of the biggest benefits of using libvirt is that it works with many different hypervisors, making it easier to manage virtualization technologies in a consistent way.

4. Prerequires for Virtualization

Before you can start virtualizing machines on your server, you need to make sure that your server meets certain prerequisites. The first and foremost requirement is that your CPU must support virtualization. To check if your CPU supports virtualization, you can run the following command in the terminal:

```
egrep -c '(svm|vmx)' /proc/cpuinfo
```

This command searches for the virtualization extensions in your CPU and returns a count of the number of matches found. If the output is greater than 0, it means that your CPU supports virtualization. The `svm` extension is for AMD processors, and `vmx` is for Intel processors.

If your CPU does not support virtualization, you will not be able to run virtual machines on your server. In this case, you may need to upgrade your CPU or consider using a different server that supports virtualization.

Once you have confirmed that your CPU supports virtualization, you can move on to the next prerequisite.

After checking if the CPU supports virtualization, the next step is to make sure that the KVM module is loaded. The KVM module is responsible for providing the virtualization functionality to the Linux kernel.

To check if the KVM module is loaded, you can use the "lsmod" command. This command lists all the currently loaded kernel modules, including the KVM module. Here's the command to check if the KVM module is loaded:

```
lsmod | grep kvm
```

If the output shows the KVM module, then it's loaded and ready to use. If it's not listed, you need to load it using the following command:

```
modprobe kvm
```

After loading the KVM module, you also need to check if the `/dev/kvm` directory exists. This directory is created by the KVM module and is used by the virtualization software to access the virtualization features provided by the CPU. You can check if this directory exists by running the following command:

```
ls /dev/kvm
```

If the directory exists, then the KVM module is working correctly, and you can proceed with virtualizing your machines. If the directory doesn't exist, you need to load the KVM module as described above or check if the hardware supports virtualization.

To check if libvirt is installed:

```
rpm -qa libvirt
```

This command will search for the libvirt package and return its version number if it is installed. If the package is not installed, you can install it using the following command:

```
sudo dnf install libvirt
```

To check if the libvirt service is running:

```
systemctl status libvirtd
```

This command will display the status of the libvirt service. If the service is not running, you can start it using the following command:

```
sudo systemctl start libvirtd
```

The libvirt library is an important component of the virtualization stack as it provides an abstraction layer that enables management of multiple hypervisors using a unified API. It supports a wide range of hypervisors including KVM, QEMU, Xen, and others, allowing virtualization administrators to choose the best hypervisor for their specific needs.

```
virt-host-validate
```

"virt-host-validate" is a Linux command that allows you to check if your system meets the prerequisites for hosting virtual machines. It is a useful tool for anyone who wants to set up a virtualized environment on their Linux machine.

The command runs a series of checks to determine if your system's hardware and software are compatible with virtualization. For example, it checks if your CPU supports hardware virtualization (SVM or VT-x), if your BIOS or UEFI settings have virtualization extensions enabled, and if the IOMMU is enabled in the kernel.

If any of the checks fail, the command will give you a detailed explanation of the issue and what you need to do to resolve it. For example, it may suggest that you update your BIOS or UEFI settings to enable virtualization, or that you install specific kernel modules to enable the IOMMU.

Overall, "virt-host-validate" is a powerful tool for anyone looking to set up a virtualized environment on their Linux machine. It can save you a lot of time and hassle by identifying any potential issues upfront, so you can address them before setting up your virtual machines.

here's another example of the `virt-host-validate` command output:

```
vbnetCopy code$ virt-host-validate

QEMU: Checking for hardware virtualization
      : PASS
QEMU: Checking if device /dev/kvm exists
      : PASS
QEMU: Checking if device /dev/vhost-net exists
      : PASS
QEMU: Checking if device /dev/net/tun exists
      : PASS
QEMU: Checking for cgroup 'cpu' controller support
      : PASS
QEMU: Checking for cgroup 'cpuacct' controller support
      : PASS
QEMU: Checking for cgroup 'cpuset' controller support
        : FAIL (No such file or directory)
QEMU: Checking for cgroup 'memory' controller support
       : PASS
QEMU: Checking for cgroup 'devices' controller support
      : PASS
QEMU: Checking for cgroup 'blkio' controller support
       : PASS
QEMU: Checking for device assignment IOMMU support
      : PASS
```

```
QEMU: Checking for OpenGL support
       : WARN (Libvirt and QEMU successfully found, but
virtualization extensions are not enabled in BIOS/UEFI)
```

This output shows that the system meets most of the prerequisites for virtualization, with only one failure and one warning. The specific checks performed include verifying the presence of hardware virtualization, checking for the necessary kernel devices, and ensuring the presence of various cgroup controllers.

In this example, the check for the 'cpuset' controller support has failed, indicating that the system does not have support for this particular cgroup controller. This could be due to a missing kernel module or a misconfiguration. The output also shows a warning that virtualization extensions are not enabled in the BIOS/UEFI settings, which is required for optimal performance.

Overall, the `virt-host-validate` command output provides a detailed overview of the system's virtualization capabilities and identifies any potential issues that need to be resolved before setting up virtual machines.

Let's go through the output of the `virt-host-validate` command step by step:

```
QEMU: Checking for hardware virtualization
      : PASS
```

This line confirms that the system has hardware virtualization support, which is a necessary prerequisite for running virtual machines.

```
QEMU: Checking if device /dev/kvm exists
      : PASS
QEMU: Checking if device /dev/vhost-net exists
      : PASS
QEMU: Checking if device /dev/net/tun exists
      : PASS
```

These lines verify the presence of the necessary kernel devices required for virtualization.

```
QEMU: Checking for cgroup 'cpu' controller support
      : PASS
QEMU: Checking for cgroup 'cpuacct' controller support
      : PASS
QEMU: Checking for cgroup 'cpuset' controller support
        : FAIL (No such file or directory)
QEMU: Checking for cgroup 'memory' controller support
       : PASS
QEMU: Checking for cgroup 'devices' controller support
      : PASS
QEMU: Checking for cgroup 'blkio' controller support
       : PASS
```

These lines confirm the presence of various cgroup controllers that are used to manage resource allocation for virtual machines. In this example, the `cpuset` controller support check has failed, indicating that the system doesn't have support for this particular cgroup controller.

```
QEMU: Checking for device assignment IOMMU support
      : PASS
```

This line checks for IOMMU support, which is a necessary prerequisite for assigning devices directly to virtual machines. The output shows that the system does support device assignment using IOMMU.

```
QEMU: Checking for OpenGL support
       : WARN (Libvirt and QEMU successfully found, but
virtualization extensions are not enabled in BIOS/UEFI)
```

Finally, this line checks for OpenGL support, which is useful for running graphical applications inside virtual machines. In this case, the output shows a warning indicating that virtualization extensions are not enabled in the BIOS/UEFI settings, which is necessary for optimal performance.

Overall, the output of the `virt-host-validate` command provides a detailed report on the system's virtualization capabilities, highlighting any issues or missing prerequisites that need to be resolved before running virtual machines.

Virt tools Packages to be installed:

- To check if VMM is installed, you can run the following command:

```
sudo dnf list virt-manager
```

- If it is not installed, you can install it with the following command:

```
sudo dnf install virt-manager
```

- To check if `virsh` is installed, you can run:

```
sudo dnf list libvirt-clientb
```

- If it is not installed, you can install it with the following command:

```
sudo dnf install libvirt-client
```

- To check if `virt-Customize` is installed, you can run:

```
sudo dnf list libguestfs-tools
```

- If it is not installed, you can install it with the following command:

```
sudo dnf install libguestfs-tools
```

- Note that virt-install is not included in the libguestfs-tools package. To install it, you can run:

```
sudo dnf install virt-install
```

- These tools are essential for creating and managing virtual machines, and you should ensure that they are installed on your server before you begin virtualizing machines.

Sure, here is a table of the virtualization tools:

Virtualization Tool	**Description**
VMM/Virt-Manager	A graphical tool for managing virtual machines
Virsh	A command-line tool for creating, managing, and monitoring virtual machines

Virtualization Tool	**Description**
Virt-Customize	A tool for customizing virtual machine images
Virt-Install	A tool for installing virtual machines
Virt-Memory-Node	A tool for monitoring and controlling virtual machine memory
Virt-P2V	A tool for converting physical machines to virtual machines
Virt-Top	A tool for monitoring virtual machines
Virt-Viewer	A graphical tool for viewing virtual machines
Virt-Manager-Remote	A tool for remotely managing virtual machines
Virt-Metrics	A tool for monitoring and collecting metrics on virtual machines

5. VM Deployment Consideration

When deploying virtual machines, there are several key considerations to keep in mind. The first consideration is the number of CPUs that you will allocate to each virtual machine. This will depend on the workload that the virtual machine will be running. If the workload is CPU-intensive, then you may need to allocate more CPUs to the virtual machine.

The second consideration is the amount of memory that you will allocate to each virtual machine. Again, this will depend on the workload that the virtual machine will be running. If the workload requires a lot of memory, then you may need to allocate more memory to the virtual machine.

The third consideration is the network. You will need to ensure that each virtual machine has access to the network, whether that be through a physical network interface or a virtual one. You may also need to consider network bandwidth, especially if the virtual machines will be communicating with each other frequently.

The fourth consideration is storage. You will need to ensure that each virtual machine has access to adequate storage, whether that be through a physical hard drive or a virtual one. You will also need to consider the speed of the storage, especially if the virtual machine will be running disk-intensive workloads.

Overall, virtual machine deployment considerations involve balancing the needs of the workload with the available resources of the host machine. It is important to carefully consider each of these factors when deploying virtual machines to ensure that they run efficiently and effectively.

6. Libvirt Defaults

Libvirt is a powerful tool that provides a variety of virtualization capabilities, including creating and managing virtual machines, storage, and networking. It comes with many default settings that can be customized according to your requirements.

The default settings for Libvirt are dependent on the hypervisor and the operating system being used. For example, when using QEMU/KVM on a Linux-based operating system, Libvirt typically uses the following default settings:

1. Storage pool: The default storage pool is usually set to /var/lib/libvirt/images. This is where the virtual machine disk images are stored by default.
2. Network: The default network is usually called "default" and is configured to use NAT (Network Address Translation). This allows the virtual machines to access the internet through the host's network connection.
3. Virtual CPUs: By default, the number of virtual CPUs allocated to a virtual machine is equal to the number of physical cores available on the host system.
4. Memory: The default amount of memory allocated to a virtual machine is usually 512 MB. This can be changed during virtual machine creation or later on using the Libvirt API.
5. Virtual NICs: The default number of virtual network interface cards (NICs) allocated to a virtual machine is usually 1. This can be changed during virtual machine creation or later on using the Libvirt API.

It is important to note that these defaults can be changed and customized according to your requirements. For example, you can create additional storage pools, configure different network types, and allocate more virtual CPUs and memory to virtual machines. The flexibility of Libvirt allows you to tailor the virtualization environment to your specific needs.

7. Libvirt Clients

Let me explain the various Libvirt Clients for creating a VM, including Virsh and its associated toolset, Virt-Manager, Cockpit, and the Libvirt API.

Virsh is a command-line tool that allows you to create, manage, and monitor virtual machines. It is part of the Libvirt toolset and uses the Libvirt API to communicate with the hypervisor. Virsh is a powerful tool that can be used to perform a wide range of virtual machine management tasks, such as creating, starting, stopping, and deleting VMs, as well as modifying their settings and viewing their status. Virsh can be used from the command line, which makes it a great tool for scripting and automation.

Virt-Manager, on the other hand, is a graphical tool that provides a user-friendly interface for managing virtual machines. It uses the Libvirt API to communicate with the hypervisor, just like Virsh. Virt-Manager is a great tool for those who prefer a graphical interface over the command line. It provides a range of features, such as creating, starting, stopping, and deleting VMs, as well as managing their settings and viewing their status.

Cockpit is a web-based management tool that allows you to manage your server from a web browser. It provides a range of features, such as monitoring system performance, managing storage, and managing virtual machines. Cockpit also uses the Libvirt API to communicate with the hypervisor, just like Virsh and Virt-Manager. Cockpit is a great tool for those who prefer to manage their server through a web-based interface.

All of these tools use the Libvirt API to communicate with the hypervisor. The Libvirt API is a powerful API that provides a wide range of functionality for managing virtual machines. However, using the Libvirt API directly can be complex and requires a good understanding of the API and the hypervisor. This is why using Virsh and its associated toolset, Virt-Manager, or Cockpit is often easier than using the Libvirt API directly. These tools provide a user-friendly interface that makes it easier to manage virtual machines, even if you don't have a lot of experience with virtualization.

8. Steps to create a virtual machine

Regardless of the specific tool you use, there are generally three main steps involved in creating a VM: downloading the image, customizing the image, and starting the VM with the image.

The first step is to download the image of the operating system that you want to use for your VM. This image is essentially a pre-built virtual hard drive that contains a fully functional OS. You can usually find these images online, either from the official website of the OS or from third-party sources.

Once you have the image downloaded, the next step is to customize it to your needs. This typically involves configuring settings such as the hostname, IP address, and installed packages. You may also need to create user accounts, set up network configurations, and install any additional software that you need.

Finally, once you have customized the image to your liking, you can start the virtual machine with the image. This is usually done using a virtualization tool such as Virsh or Virt-Manager, which allows you to select the image and create a new virtual machine using it. Once the VM is started, you can access it just like you would a physical machine, using remote desktop tools or a command-line interface.

Overall, the process of creating a virtual machine is fairly straightforward, and can be accomplished using a variety of different tools and methods. By following the steps outlined above, you can quickly and easily create a VM that meets your specific needs.

8.1. First step: Download the Image

When it comes to creating a virtual machine, the first step is to download an operating system image. There are various operating systems or distributions to choose from, depending on your needs. For example, CentOS is a popular distribution used for servers, while Ubuntu is a popular choice for desktops and servers.

To download CentOS, you can visit the official CentOS website at **https://www.centos.org/download/**. Here, you can select the desired version and architecture of CentOS that you want to download. Once you have selected the appropriate image, you can download it to your local machine.

Similarly, to download Ubuntu, you can visit the official Ubuntu website at **https://ubuntu.com/download**. Here, you can select the desired version and architecture of Ubuntu that you want to download. Once you have selected the appropriate image, you can download it to your local machine.

After downloading the operating system image, you can move on to the next step of customizing the image to fit your needs before starting the virtual machine.

8.2. Second Step: Customize the Image

There are various tools available that allow you to customize the image according to your needs. One of the most commonly used tools is "Virt-Customize". This tool allows you to customize the virtual machine image by adding packages, editing configuration files, creating users, and modifying the network settings. You can also use "Cloud-Init" to automate the customization process by specifying the settings in a YAML file.

Another popular tool for customizing images is "Packer". Packer allows you to create machine images for multiple platforms, including virtual machines. It provides a scriptable way to build images, which allows you to automate the customization process and ensure consistency across multiple images.

Lastly, you can also use a simple script or configuration management tool like Ansible, Puppet, or Chef to customize the image. These tools allow you to define the desired state of the image in a script or configuration file, and then apply that configuration to the image.

Overall, there are multiple ways to customize virtual machine images, and the choice of tool will depend on your specific needs and preferences.

Here are the detailed commands to use "virt-customize" to customize the CentOS image with a root password and hostname:

First, make sure that you have the "virt-customize" tool installed on your system. You can check if it's installed by running the following command:

```
virt-customize --version
```

If the tool is not installed, you can install it using the package manager for your distribution. Copy the CentOS image that you downloaded in the previous step to this working directory.

```
cp /path/to/centos-image.qcow2 /var/tmp/vm-customizations
```

To set the root password, use the "virt-customize" tool with the "--root-password password:PASSWORD" option. For example, to set the root password to "mypassword", run the following command:

```
virt-customize --root-password password:mypassword -a
/var/lib/libvirt/images/centos.qcow2
```

This command sets the root password to "mypassword" for the CentOS image located at "/var/lib/libvirt/images/centos.qcow2".

To set the hostname, use the "virt-customize" tool with the "--hostname NEW_HOSTNAME" option. For example, to set the hostname to "myhostname", run the following command:

```
virt-customize --hostname myhostname -a
/var/lib/libvirt/images/centos.qcow2
```

This command sets the hostname to "myhostname" for the CentOS image located at "/var/lib/libvirt/images/centos.qcow2".

Once you have customized the image with the desired settings, you can use it to create a virtual machine. In the next section, we will discuss how to create a virtual machine using the customised image.

8.3. Third Step: Create the VM

Creating a virtual machine (VM) is the final step in the process of deploying a virtual environment. The VM is the guest operating system that runs within the virtualization environment. There are several tools available to create a VM, including VMM (Virtual Machine Manager) and Virt-Install.

VMM is a graphical tool that provides a user-friendly interface for creating and managing virtual machines. It allows you to easily create new VMs and manage existing ones. You can also monitor the performance of VMs and make changes to their configuration.

On the other hand, Virt-Install is a command-line tool that provides more fine-grained control over the VM creation process. It allows you to specify the configuration options for the VM using command-line arguments, or by using an XML file. This is useful if you need to automate the VM creation process, or if you need to create VMs in bulk.

To create a VM using VMM, you can follow these steps:

1. Open VMM on your host machine.
2. Click on the "Create a new virtual machine" button.
3. Choose the installation method for the VM (for example, by installing from a local ISO file or over the network).
4. Configure the VM's hardware resources such as CPU, memory, storage, and network settings.
5. Customize any additional options as needed.
6. Click on "Finish" to create the VM.

To create a VM using Virt-Install, you can use the following command as an example:

Here is the reformatted command with each flag in a new line:

```
cssCopy codevirt-install \
--name myvm \
--ram 2048 \
--vcpus 2 \
--disk path=/var/lib/libvirt/images/myvm.qcow2,size=10 \
--os-type linux \
--os-variant centos7.0 \
--location /path/to/iso \
--network network=default \
--graphics vnc \
--console pty,target_type=serial \
--extra-args='console=ttyS0,115200n8 serial'
```

This command is used to create a virtual machine using the `virt-install` tool. Here is a brief explanation of each flag:

- `--name myvm`: specifies the name of the virtual machine as `myvm`.
- `--ram 2048`: sets the amount of memory allocated to the virtual machine to 2048 MB.
- `--vcpus 2`: sets the number of virtual CPUs allocated to the virtual machine to 2.
- `--disk path=/var/lib/libvirt/images/myvm.qcow2,size=10`: specifies the path and size of the disk image for the virtual machine.
- `--os-type linux`: sets the operating system type to Linux.
- `--os-variant centos7.0`: sets the operating system variant to CentOS 7.0.
- `--location /path/to/iso`: specifies the location of the installation ISO image.
- `--network network=default`: connects the virtual machine to the default network.
- `--graphics vnc`: enables VNC graphics for the virtual machine.
- `--console pty,target_type=serial`: sets up a serial console for the virtual machine.
- `--extra-args='console=ttyS0,115200n8 serial'`: specifies additional kernel boot parameters.

This command creates a VM with the name "myvm", 2 vCPUs, 2GB of RAM, and a 10GB disk image. It uses the CentOS 7.0 operating system, and installs it from the ISO located at "/path/to/iso". It also configures the VM's network settings to use the default network, and enables VNC and serial console access.

Another use:

```
virt-install \
--name [vm-name] \
--memory [memory-size] \
--vcpus [number-of-virtual-cpus] \
--disk path=[path-to-disk-image],format=[disk-format] \
--cdrom [path-to-iso-file] \
--os-type [operating-system-type] \
--network bridge=[bridge-name] \
--graphics [graphics-type] \
--console pty,target_type=serial \
--noautoconsole
```

where `[vm-name]` is the name that you want to give to the new virtual machine, `[memory-size]` is the amount of memory that you want to allocate to the virtual machine, `[number-of-virtual-cpus]` is the number of virtual CPUs that you want to assign to the virtual machine, `[path-to-disk-image]` is the path to the disk image file, `[disk-format]` is the format of the disk image (e.g. qcow2, raw), `[path-to-iso-file]` is the path to the ISO file that contains the operating system that you want to install, `[operating-system-type]` is the type of operating system that you want to install (e.g. linux), `[bridge-name]` is the name of the network bridge that the virtual machine will use to connect to the network, `[graphics-type]` is the type of graphics that the virtual machine will use (e.g. vnc), and `--noautoconsole` disables the automatic console connection on VM start.

In summary, creating a VM involves choosing the right tool for the job, configuring the hardware resources and settings, and finally customizing the guest operating system to meet your needs.

9. Managing VM

Managing a virtual machine involves performing various tasks such as starting, stopping, monitoring, and configuring the virtual machine. There are several tools available for managing virtual machines, including VMM and Virsh.

VMM, or Virtual Machine Manager, is a graphical tool that allows users to manage virtual machines using a simple and intuitive interface. With VMM, users can easily create, edit, start, stop, and monitor virtual machines. VMM also provides a detailed view of virtual machine performance and resource usage, making it easier to troubleshoot issues and optimize virtual machine performance.

On the other hand, Virsh is a command-line tool that provides a powerful and flexible way to manage virtual machines. With Virsh, users can perform almost any operation on a virtual machine, from starting and stopping the virtual machine to configuring and monitoring its performance. Virsh provides a rich set of commands and options that can be used to automate complex tasks and customize virtual machine configurations.

Both VMM and Virsh use the libvirt API to communicate with the virtualization hypervisor. The libvirt API provides a common interface for managing virtual machines, regardless of the underlying virtualization technology. This makes it easier to manage virtual machines across different hypervisors and platforms.

Using VMM or Virsh to manage virtual machines is typically easier than using the libvirt API directly because these tools provide a simpler and more user-friendly interface. VMM is particularly useful for users who prefer a graphical interface, while Virsh is ideal for users who prefer command-line tools and need more flexibility and automation capabilities.

With VMM, you can easily manage virtual machines with a graphical user interface. Here are the basic steps for editing, starting, stopping, and monitoring virtual machines:

1. Open VMM and select the virtual machine you want to manage.
2. To edit the virtual machine settings, click the "Edit" button. This will bring up a dialog where you can change settings such as the virtual hardware, network configuration, and storage devices.

3. To start the virtual machine, select it and click the "Start" button. The virtual machine will start up and begin running.
4. To stop the virtual machine, select it and click the "Stop" button. This will initiate a graceful shutdown of the virtual machine, allowing it to save any data and cleanly shut down.
5. To monitor the virtual machine, select it and click the "Monitor" button. This will open a console window where you can view the virtual machine's operating system and interact with it as if you were sitting at the physical console.

In addition to these basic operations, VMM also provides more advanced features such as live migration, cloning, and snapshotting of virtual machines. With these features, you can easily move virtual machines between hosts, create copies of existing virtual machines, and save snapshots of virtual machines at specific points in time for backup or testing purposes.

Here are the steps to manage a virtual machine using virsh:

1. To list all the available virtual machines, use the following command:

```
virsh list --all
```

1. To start a virtual machine, use the following command:

```
virsh start [vm-name]
```

where `[vm-name]` is the name of the virtual machine that you want to start.

1. To stop a running virtual machine, use the following command:

```
virsh shutdown [vm-name]
```

where `[vm-name]` is the name of the virtual machine that you want to stop.

1. To force stop a virtual machine, use the following command:

```
virsh destroy [vm-name]
```

where `[vm-name]` is the name of the virtual machine that you want to force stop.

1. To check the status of a virtual machine, use the following command:

```
virsh domstate [vm-name]
```

where `[vm-name]` is the name of the virtual machine that you want to check.

1. To connect to the console of a virtual machine, use the following command:

```
virsh console [vm-name]
```

where `[vm-name]` is the name of the virtual machine that you want to connect to.

1. To change the configuration of a virtual machine, use the following command:

```
virsh edit [vm-name]
```

These are some of the basic commands that you can use to manage virtual machines using virsh.

10. Managing the Network

When managing virtual machines, it is also important to consider the network configuration. Both VMM and Virsh allow for easy management of the network settings for your virtual machines. With VMM, you can create, edit, and delete virtual networks, as well as assign network interfaces to virtual machines. To manage network settings for a virtual machine in VMM, follow these steps:

1. Open VMM and select the virtual machine you want to manage.
2. Click on the "Network Interfaces" tab.
3. From here, you can add or remove network interfaces and assign them to virtual networks.

With Virsh, you can manage virtual networks and their settings using the "virsh net" command. To manage network settings for a virtual machine using Virsh, follow these steps:

1. Open a terminal and enter the command "virsh net-list" to view a list of available virtual networks.
2. Enter the command "virsh net-edit " to edit the network settings.
3. From here, you can configure the network settings, including the subnet, IP address range, and DNS settings.

Overall, both VMM and Virsh provide easy and efficient ways to manage the network settings for your virtual machines.

11. Managing the Storage

Managing storage in virtual machines can be done using either VMM or Virsh. Both tools offer different ways to manage storage, and you can choose the one that best suits your needs.

With VMM, you can manage storage by creating storage pools and adding storage volumes to them. To create a new storage pool, you can follow these steps:

1. Open VMM and navigate to the "Storage" tab.
2. Click on "New Storage Pool" and enter a name for the storage pool.
3. Select the type of storage pool you want to create. You can choose between different options like file-based storage, block-based storage, or network storage.
4. Specify the location where you want to store the storage pool.
5. Choose the target path and options for the storage pool.

Once you have created the storage pool, you can add storage volumes to it. Here's how:

1. Navigate to the "Storage" tab in VMM.
2. Click on the storage pool you want to add a volume to.
3. Click on "New Volume" and specify the name and size of the volume.
4. Choose the format for the volume. You can choose between different options like raw or qcow2.
5. Specify the target path and options for the volume.

With Virsh, you can manage storage by creating disk images and attaching them to virtual machines. To create a new disk image, you can follow these steps:

Open a terminal window and type the following command:

```
virsh vol-create-as [pool-name] [volume-name] [size]
```

Replace [pool-name] with the name of the storage pool you want to use, [volume-name] with the name you want to give to the volume, and [size] with the size of the volume you want to create.

Once the volume has been created, you can attach it to a virtual machine by typing the following command:

```
virsh attach-disk [vm-name] [disk-path] [disk-device]
```

Replace [vm-name] with the name of the virtual machine you want to attach the disk to, [disk-path] with the path to the disk image you created earlier, and [disk-device] with the device name you want to use for the disk.

Overall, both VMM and Virsh offer powerful ways to manage storage in virtual machines, and you can choose the one that best suits your needs.

12. Conclusion

In conclusion, server virtualization has become an essential tool for businesses and individuals alike in optimizing resource utilization, enhancing flexibility, and reducing costs. In this book, we have explored the basics of server virtualization, including its definition, benefits, and history. We have also delved into the various components that make up server virtualization, such as the hardware, hypervisor, and virtual machines.

Furthermore, we have examined the challenges that arise in CPU virtualization and the solutions that have been developed to overcome them. We have also explored the virtualization stack using Qemu, KVM, libvirt, and virt tools, as well as VM deployment considerations, libvirt defaults, and steps to create a virtual machine.

Throughout this book, we have emphasized the importance of understanding the different components of server virtualization and the best practices for managing them. By mastering these techniques and tools, you can achieve optimal performance and cost savings for your business or personal needs.

As Alan Watts once said, "The only way to make sense out of change is to plunge into it, move with it, and join the dance." We hope this book has given you the knowledge and confidence to join the dance of server virtualization and reap its benefits.