Lesson 1: The Foundations, History, and Core Benefits of Virtualization

Virtualization is a method of abstracting the physical hardware of a single computer into several different execution environments. It is conceptually similar to a "layered approach" in operating systems, but here, the layer creates a full virtual system—the **Virtual Machine (VM)**—on which entire operating systems and applications can run.

1. The Three Primary Components

Every virtualized system is made up of these three parts:

- **The Host:** The underlying physical hardware system (the actual server, CPU, memory, and storage).
- **The Virtual Machine Manager (VMM) or Hypervisor:** The software or firmware layer that creates and runs VMs. Its main job is to provide an interface to the guest that is essentially **identical** to the host's hardware.
    - *Important Exception:* In **paravirtualization**, this interface is intentionally modified to help the guest and VMM work together more efficiently.
- **The Guest:** A process provided with a virtual copy of the host hardware. This guest is almost always a full Operating System.

2. The History and "True" Definition

Virtualization isn't new; it has evolved over several decades:

- **IBM Mainframes (1972):** Virtualization first appeared to allow multiple users to share a single "batch-oriented" mainframe system at once.
- **The Three Requirements for Virtualization:** To move beyond just IBM systems, researchers created a formal definition. A VMM must meet these three criteria:
    1. **Isomorphism:** It must provide an environment for programs that is essentially identical to the original physical machine.
    2. **Efficiency:** Programs running in the VM should only show minor performance decreases.
    3. **Resource Control:** The VMM must remain in complete control of all system resources at all times.
- **The PC Revolution (Late 1990s):** Intel CPUs finally became fast enough for researchers to try virtualization on standard personal computers. This led to the creation of companies like **VMware** and **Xen**, whose technologies are still standard today.

3. Core Benefits and Features

Using VMs provides several massive advantages over running software directly on physical "bare metal" hardware:

- **Security and Protection:** The host system is protected from the VMs, and the VMs are isolated from each other. If a VM is infected by a virus, it is much less likely to spread to the host or other guests. Although VMs are isolated for protection, they can still communicate or share data through shared file system volumes or standard network communication.
- **Consolidation:** You can run many different operating systems on one physical machine. This is highly efficient for datacenters and application development.
- **State Management:**
    - **Freeze/Suspend:** You can pause a running VM, move it to another system, and resume it.
    - **Snapshots:** You can save the exact "state" of a VM at any moment. If you make a mistake later, you can restore the VM back to that specific snapshot instantly. Some managers allow multiple snapshots per VM.
- **Cloning:** You can create an exact copy of a VM and run both the original and the copy simultaneously.
- **Templating:** You can set up a "perfect" VM with an OS and specific apps, then use that "template" to quickly create multiple identical instances for customers or tasks.
- **Live Migration:** This advanced feature allows you to move a running VM from one physical host to another without any interruption to the user's access.
- **The Path to Cloud Computing:** When you take **Templating** and **Live Migration** together then add **APIs** (which allow programs to automatically tell the infrastructure to create new VMs or virtual desktops), you get **Cloud Computing** that allow programs to automatically create new guests and virtual desktops.

4. The Technical "Building Blocks"

Implementing a VM is technically complex because the guest must believe it is in control of the hardware:

- **The Virtual CPU (VCPU):** Most VMMs implement a VCPU to represent the state of the CPU as the guest believes it to be. When the VMM switches a guest onto the physical CPU, it loads the information from this VCPU into the real hardware.
- **Hardware Challenges:** It was historically difficult to provide an exact duplicate of a machine if the CPU only supported **dual-mode operation** (User vs. Kernel mode). Virtualization has become much easier as modern CPUs have added specific hardware features to support VMMs.

- **Lifecycle Management:**
    - **Creation:** The VMM creates the VM and assigns it specific resources like CPU cores, memory totals, and networking/storage details.
    - **Deletion:** When the VM is no longer needed, it is deleted, and its resources are immediately freed for other uses.
    - **Speed:** These steps are significantly faster and simpler than installing a physical machine.
    - **The Sprawl Risk:** Because it is so easy to create VMs, organizations can suffer from **"virtual machine sprawl,"** where they have so many VMs that it becomes difficult to track their history, purpose, and current state.

Lesson 2: The Three Major Hypervisor Types (0, 1, and 2)

In this lesson, we will examine how Virtual Machine Managers (VMMs), also known as hypervisors, are actually implemented. There is a great deal of variation in how these systems are built, and they are generally categorized into three types based on where they sit in relation to the physical hardware and the operating system.

1. Type 0 Hypervisors: The Hardware-Based Approach

Type 0 hypervisors are an older concept, often referred to by hardware manufacturers as "partitions" or "domains". Type 0 generally has a **smaller feature set** than the other types because the logic is embedded in firmware

- **Implementation:** This is a hardware feature implemented directly by the system's firmware. Because the virtualization logic is embedded so deeply, the operating system does not need to do anything special to function as a guest.
- **Resource Allocation:** In a Type 0 environment, resources like CPUs and memory are usually dedicated to each guest rather than shared.
- **The I/O Challenge:** Managing Input/Output (I/O) is difficult here because it is hard to provide enough physical devices and controllers to dedicate a unique set to every single guest. To solve this, a VMM might implement a "control partition" running background processes (daemons) that other guests communicate with to share I/O resources.
- **Unique Capability:** Type 0 hypervisors are particularly good at "virtualization-within-virtualization," meaning a guest running on a Type 0 hypervisor can itself be a VMM with its own guests, a feat other types struggle to achieve.
- **Examples:** Common examples include IBM LPARs and Oracle LDOMs.

2. Type 1 Hypervisors: The "Datacenter Operating System"

Type 1 hypervisors are special-purpose operating systems designed specifically to create, run, and manage guest operating systems natively on the host hardware.

- **Core Functionality:** They run in kernel mode and must implement their own device drivers for the host hardware because no other component exists to handle them. Guests typically have no idea they are running inside a virtual machine.
- **Usage:** These are commonly found in company datacenters because they allow managers to consolidate multiple OSes onto less hardware and move guests between systems to balance performance.
- **The General-Purpose Variation:** Some general-purpose operating systems, like Microsoft Windows Server with Hyper-V or RedHat Linux with KVM, also provide Type 1 VMM functionality. While they perform standard OS duties alongside VMM duties, they are typically less feature-rich than dedicated, "pure" Type 1 hypervisors. However, it requires special handling whenever that guest tries to execute "special instructions" that would normally affect the whole system.
- **Type 1 Nesting:** A Type 1 hypervisor can actually run on top of a Type 0 hypervisor, but it cannot run on another Type 1 hypervisor.
- **Examples:** Key examples include VMware ESX, Joyent SmartOS, and Citrix XenServer.

3. Type 2 Hypervisors: The Application-Based Approach

Type 2 hypervisors are less interesting from a pure OS design perspective because there is very little OS involvement in the virtualization process itself.

- **Implementation:** The VMM is simply an application that runs on a standard operating system like Windows, Linux, or macOS. The host operating system treats the VMM like any other process and often does not even know it is hosting guests.
- **Performance Trade-offs:** Because they run on top of another OS, Type 2 hypervisors generally have poorer performance and cannot take full advantage of certain hardware features.
- **The Benefit for Users:** The primary advantage is that they require no changes to the host OS, making them perfect for students or developers who want to run multiple different guest OSes on their personal laptop.
- **Examples:** Famous examples include VMware Workstation, VMware Fusion, Oracle VirtualBox, and Parallels Desktop.

Comparison and Summary Table

Feature	Type 0	Type 1	Type 2
Location	Firmware	Native Hardware (Bare Metal)	Application on Host OS
I/O Handling	Dedicated HW or Control Partition	VMM-implemented drivers	Managed by Host OS
Best For	High-end server partitions	Datacenters and Consolidation	Individual/Student use
Guest Awareness	None (OS is unchanged)	Generally none	None

Lesson 3: Specialized Implementation Methods

In this lesson, we explore specialized techniques that deviate from standard virtualization to solve specific problems like performance bottlenecks, hardware incompatibility, or the need for isolated application environments.

1. Paravirtualization: Cooperative Optimization

Paravirtualization technically breaks the traditional definition of virtualization because the VMM does not present an exact duplicate of the underlying hardware. Instead, the guest operating system is modified to work in cooperation with the VMM to optimize performance.

- **Xen Implementation:** Xen is the leader in this space and introduced several key techniques:
    - **Efficient I/O via Shared Circular Buffers:** To handle communication, Xen uses a circular buffer in shared memory. This follows a **producer-consumer model** where the guest and VMM use private and shared pointers to manage a Request queue (from the VM) and a Response queue (from the VMM).
    - **Memory Management via Hypercalls:** Instead of standard page tables, guests have their own read-only tables. When a change is needed, the guest issues a **hypercall** (a direct call to the hypervisor) to perform the update.
- **Historical Context:** This method allowed for the virtualization of older x86 CPUs without needing complex "binary translation".
- **The Modern Shift:** Because modern CPUs now have built-in hardware support for virtualization, guest modification is often no longer required. When running an unmodified guest on modern hardware, Xen is no longer performing paravirtualization.

2. Emulation: Bridging Different Architectures

Emulation is used when the host system's hardware (specifically the CPU architecture) is completely different from what the guest OS was compiled for.

- **Instruction Translation:** The emulator must translate every single guest instruction into a format the native CPU can understand.
- **The Performance Penalty:** Because of this constant translation, emulation is typically an **order of magnitude slower** than native code.
- **Use Cases:** It is essential for running legacy applications on newer, incompatible hardware. It is also extremely popular in the gaming industry for running old console games on modern systems.

3. Application Containment (Containers)

This is not considered "full" virtualization because there is only one kernel (the host OS) running for the entire system. It provides virtualization-like features by segregating applications from the operating system.

- **Virtual Layering:** Systems like **Oracle Solaris Zones**, **BSD Jails**, and **IBM AIX WPARs** create a virtual layer between the OS and the apps.
- **Zone Isolation:** Each zone or container acts like an independent system, provided with its own:
    - Networking stack (unique IP addresses, ports).
    - User accounts.
    - Applications and file access.
- **Resource Allocation:** Physical CPU and memory are divided between zones. A specific zone can even have its own **internal scheduler** to manage the resources it has been assigned.

4. Programming-Environment Virtualization (The JVM)

This method does not virtualize real hardware but instead creates an optimized, software-based virtual system for a specific language environment. In this context, virtualization is defined as providing APIs that offer a set of features to a language and its programs to create an improved execution environment.

The premier example is the **Java Virtual Machine (JVM)**, which was invented by Sun Microsystems in 1995 with the goal of "Write once, run anywhere".

- **Components:** The environment includes the Java language specification, an API library, and the JVM itself.
- **The Execution Process:**
    - **Bytecode:** Java objects are defined using a **class construct**. A Java program, consisting of one or more objects, is compiled into **architecture-neutral bytecode** (.class files) rather than native machine code.
    - **Class Loader:** This component pulls in the program's .class files along with any required **Java API .class files**.
    - **Interpreter:** The JVM executes this bytecode on the local host system.

- **Portability:** To achieve portability, the JVM itself is compiled specifically for each host architecture (such as Windows, Linux, or even smart phones). This allows a Java program to run in the JVM regardless of the underlying hardware.
- **Advanced Optimization Features:**
    - **Garbage Collection:** The JVM automatically manages memory by reclaiming space from objects that are no longer in use.
    - **Just-In-Time (JIT) Compiler:** To increase speed, the JIT compiler identifies frequently used bytecodes, translates them into native machine code, and caches them for faster execution in the future.
- **Other Examples: Microsoft .Net** is another major example of this type of virtualization.

This completes the detailed look at the JVM as a specialized implementation method. Since this was for Lesson 3, are you ready to see how the VMM manages resources like CPU and Memory in **Lesson 4**?

Lesson 4: Managing Hardware: CPU, Memory, and Storage in Virtual Environments

In this lesson, we examine how Virtual Machine Managers (VMMs) act as sophisticated traffic controllers to manage the physical resources of the host machine. When many guests run simultaneously, the VMM must ensure each gets the processing power, memory, and storage it requires, even when those resources are oversubscribed.

1. CPU Scheduling and the Virtual CPU (VCPU)

Even a system with a single physical CPU can act like a multiprocessor system because the VMM can present one or more **Virtual CPUs (VCPUs)** to every guest.

- **VCPU Management:** The VMM creates a VCPU to represent the processor state as the guest OS expects it; the number of VCPUs assigned to a VM can be adjusted throughout its lifecycle.
- **CPU Overcommitment:** When there aren't enough physical cores for every guest, the VMM uses standard scheduling algorithms—often with an added **fairness aspect**—to decide which guest threads get to use the real hardware.
- **The "Cycle Stealing" Problem:** Because the VMM must "steal" cycles for its own management tasks and other guests, the Guest OS may not receive the cycles it expects. For example, a 100ms time slice might take one second or longer in real-time, leading to poor user response and incorrect time-of-day clocks.
- **Timing Fixes:** Many VMMs provide a specialized application or driver to run inside the guest to synchronize the clock and provide better integration with the host.

2. Advanced Memory Management

Memory management is uniquely challenging because guests often require large amounts of RAM that frequently exceed the host's physical capacity. To handle this **memory oversubscription**, VMMs like VMware ESX use three primary methods:

- **Double-Paging:** This happens when a guest's own page table thinks a page is in a physical frame, but the VMM has secretly moved it to a **backing store** on a disk to free up RAM.
- **The Balloon Memory Manager:** The VMM installs a **pseudo-device driver** that adds kernel-mode code to the guest OS. When the host is low on memory, it tells this "balloon" to inflate (allocate memory within the guest), forcing the guest's own kernel to free up physical RAM for the VMM to reclaim.
- **Deduplication:** The VMM identifies identical memory pages across different guests (such as the same OS kernel code) and maps them to a single physical frame to save space.

3. I/O and Networking Complexity

I/O is often the most complicated part of virtualization because of hardware variety and the need for high-speed, optimized paths.

- **I/O Optimization:** To minimize hypervisor interference, modern VMMs use **Direct Device Access**, **DMA Pass-through**, or **Direct Interrupt Delivery**, though these require specific hardware support.
- **Networking Models:** There are two main ways a VMM provides network access:
    - **Bridging:** The VMM bridges the guest directly to the physical network, making it appear as an independent machine.
    - **Network Address Translation (NAT):** The VMM provides a local address and translates it to the host's address for external communication, effectively hiding the guest's identity.

4. Storage Management and Disk Images

VMMs must provide guests with both boot disks and data storage, which requires more flexibility than standard disk partitioning can provide.

- **Storage Formats:** Type 1 hypervisors store guest root disks as **disk images** within their own file system, while Type 2 hypervisors store them as standard files in the host's file system.
- **Portability:** Because these disks are just files, you can create a new guest by duplicating a file or move a guest to a different server just by moving its image.
- **Conversions:** VMMs use **Physical-to-Virtual (P-to-V)** tools to convert a physical disk into a virtual format, and **Virtual-to-Physical (V-to-P)** tools to convert it back for use on native hardware.

- **Broad Support:** Beyond disk images, VMMs provide access to **Network Attached Storage (NAS)**, raw physical disks, and specific partitions.

Lesson 5: Live Migration and Real-World Examples

In this final lesson, we will look at one of the most advanced features of virtualization—**Live Migration**—and then examine a specific real-world examples: **VMware**.

1. Live Migration: Moving VMs Without Downtime

Live migration is a unique feature of virtual machine managers that allows you to move a running guest from one physical host to another without interrupting the user's access to the guest or its applications. This is incredibly useful for balancing server loads or performing hardware maintenance without turning off the system.

The 7-Step Migration Process: To ensure a smooth transition, the VMM follows these specific steps:

1. **Establish Connection:** The source VMM makes a connection with the target VMM on the new server.
2. **Target Preparation:** The target VMM creates a new guest "container" by setting up a new VCPU and other necessary structures.
3. **Read-Only Transfer:** The source sends all of the guest’s read-only memory pages to the target.
4. **Initial Read-Write Transfer:** The source sends all read-write memory pages to the target and marks them as "clean."
5. **Iterative Transfer:** While step 4 is happening, the guest is still running and might change some pages (making them "dirty"). The source repeats the transfer of these modified pages.
6. **The Freeze and Handover:** When the transfer cycle becomes very short, the source VMM freezes the guest, sends the final VCPU state, the last few dirty pages, and tells the target to start running the guest.
7. **Termination:** Once the target acknowledges that the guest is running, the source terminates the original guest instance.

2. Real-World Example: VMware Workstation (Type 2 Hypervisor)

- **Architecture:** It runs as an application on top of a standard host operating system like Windows or Linux.
- **Hardware Abstraction:** It provides the guest with its own virtual CPUs, memory, network interfaces, and disk drives.
- **Storage Flexibility:** Guests can be given access to real physical disks, or more commonly, virtual disks which are simply files stored within the host's file system. This makes them easy to move or duplicate.