Virtualization

Introdução Engenharia Informática

Mário Antunes

October 06, 2025

Universidade de Aveiro

Running a Computer Inside Your Computer

The Core Idea: What is Virtualization?

Virtualization creates a software-based, or “virtual,” version of a computer. This Virtual Machine (VM) runs as an application on your physical computer but behaves like a completely separate machine.

- **Host:** Your physical machine and its Operating System (OS).
- **Guest:** The virtual machine and the OS it runs.
- **Hypervisor:** The software that creates and manages the VMs.

The Challenge: Managing Privileged Instructions

A normal application can't access hardware directly; it must ask the Host OS. But a Guest OS *expects* to have full control. How do we solve this conflict safely?

The hypervisor's main job is to intercept and safely manage the guest's requests for privileged hardware access. The way it does this defines the difference between emulation and virtualization.

Emulation: Definition & Use Case

Definition: Emulation involves using software to mimic the hardware of a *different* system. The hypervisor acts as a translator, converting instructions from the guest's CPU architecture to the host's CPU architecture.

Use Case Example: Running a classic video game designed for an ARM-based console (like the Nintendo Switch) or a PowerPC-based console (like the GameCube) on your x86-based PC. The emulator (e.g., Yuzu or Dolphin) translates the game's code in real-time.

Emulation: Path of an Instruction

The hypervisor (emulator) must inspect and translate every instruction in software before it can be executed by the host hardware.

Emulation: Advantages & Disadvantages

Advantages 👍

- **Cross-Architecture Compatibility:** Its greatest strength. It allows software designed for one type of CPU (e.g., ARM) to run on a completely different type (e.g., x86).

Disadvantages 👎

- **Very Slow:** The software translation step for every instruction creates significant performance overhead, making it much slower than running native code.
- **High Resource Usage:** The translation process itself is computationally expensive and consumes a lot of host CPU cycles.

Full Virtualization: Definition & Use Case

Definition: Full Virtualization runs an *unmodified* guest OS on a simulated hardware environment that matches the host's architecture. It relies on **CPU hardware assistance** (Intel VT-x / AMD-V) to run code efficiently. The guest OS is completely unaware that it's being virtualized.

Use Case Example: A macOS user running a full version of Windows 11 in VirtualBox to use a specific piece of software that is not available on macOS, like a CAD program or a particular PC game.

Full Virtualization: Path of an Instruction

Non-privileged instructions run directly on the host CPU at full speed. When the guest attempts a privileged instruction, the CPU hardware automatically **traps** it and transparently hands control over to the hypervisor to handle it safely.

Full Virtualization: Advantages & Disadvantages

Advantages 👍

- **High Compatibility:** Can run any standard, off-the-shelf operating system without any changes.
- **Good Performance:** Hardware assistance makes it significantly faster than emulation.
- **Strong Isolation:** Guests are securely isolated from the host and each other by the hardware.

Disadvantages 👎

- **Trap Overhead:** The “trap-and-emulate” cycle for privileged instructions still introduces some performance overhead, which can be significant for I/O-heavy workloads.

Paravirtualization: Definition & Use Case

Definition: In Paravirtualization, the guest OS is *aware* that it's a VM and has been modified with special drivers. Instead of performing actions that would be trapped, it communicates directly with the hypervisor through an efficient API.

Use Case Example: This is the foundation of modern cloud computing. A high-performance web server running on an Amazon Web Services (AWS) EC2 instance uses paravirtualized **VirtIO** drivers for its disk and network devices to achieve maximum throughput and low latency.

Paravirtualization: Path of an Instruction

The guest OS knows it can't directly access hardware, so its aware driver makes a **"Hypercall"**—a direct and highly efficient function call—to the hypervisor, completely avoiding the trap mechanism.

Paravirtualization: Advantages & Disadvantages

Advantages 👍

- **Highest Performance:** By avoiding the trap overhead, it offers the best performance, especially for disk and network-intensive tasks.
- **Efficient:** Lower CPU overhead compared to full virtualization because the communication path is optimized.

Disadvantages 👎

- **Requires Guest OS Modification:** Cannot run an unmodified, off-the-shelf OS. The OS must have the specific paravirtualization drivers installed (though most modern OSes now include them).

Comparison Summary

Feature	Emulation	Full Virtualization	Paravirtualization
Core Concept	Mimic different hardware	Isolate an unmodified OS	Cooperate with an aware OS
Performance	Very Low	Good	Excellent
Guest OSModification	No	No	**Yes**
Hardware	Any guest on any host	Guest and Host mustshare same architecture	Guest and Host mustshare same architecture
PrimaryMechanism	Software Translation	Hardware Trap & Emulate	Hypercalls
TypicalUse Case	Retro Gaming, Cross-Architecture Dev	Desktop Use, LegacySystem Encapsulation	Cloud Computing, DataCenters, High-Perf. Servers

How PCI Passthrough is Achieved

This advanced feature requires hardware support from the CPU and motherboard chipset, specifically the **IOMMU (Input-Output Memory Management Unit)**.

- **Intel's IOMMU:** VT-d
- **AMD's IOMMU:** AMD-Vi

The IOMMU acts as a hardware-level traffic controller. It creates a secure memory sandbox for the device, ensuring that when it's passed through to a VM, it can only access the memory of that specific VM. This prevents the device from interfering with the host OS or other VMs, providing both high performance and strong security.

Use Case 1: Data Centers and Servers

Virtualization is the backbone of the modern cloud.

- **Server Consolidation:** One powerful physical server can replace dozens of older ones by running each as a separate VM, saving power, cooling, and space.
- **Snapshots & High Availability:** Instantly save and restore the state of a VM. VMs can even be migrated between physical servers with zero downtime.

The Problem: Repetitive VM Setup

Imagine you need to deploy 10 identical web server VMs. The manual process for *each one* would be:

1. Boot the VM and log in.
2. Set a unique hostname.
3. Configure the network.
4. Create user accounts and set up SSH keys.
5. Run security updates (`apt update && apt upgrade`).
6. Install necessary software (`nginx`, `ufw`, etc.).
7. Configure services.

This is slow, tedious, and prone to human error. It simply doesn't scale for cloud environments.

The Solution: Cloud-Init ☁️

Cloud-Init is the industry-standard tool for automating the **initial setup** of a cloud instance or virtual machine. It is designed to run **only on the very first boot** to provision the system.

- **How it Works:**
  1. The cloud platform or hypervisor provides configuration data (called “user data”) to the VM as it’s being created.
  2. Inside the guest OS, a Cloud-Init service starts automatically on the first boot.
  3. This service finds the user data and executes the instructions within it to configure the system.
- **Analogy:** Think of Cloud-Init as an automated setup script that configures your new server for you before you ever log in for the first time.

Cloud-Init in Practice: User Data

The configuration for Cloud-Init is typically written in a simple text format called **YAML**. This file, often named `user-data`, contains a set of directives. With this single file, a new VM can boot up as a fully configured and ready-to-use web server, with no manual intervention required.

Here's a practical example of a `user-data` file that sets up a basic web server:

```yaml
#cloud-config
# Set the hostname for the server
hostname: webserver-01

# Create a new user named 'admin', give them sudo rights, and add an SSH key
users:
  - name: admin
    sudo: ALL=(ALL) NOPASSWD:ALL
    groups: sudo
    shell: /bin/bash
    ssh_authorized_keys:
      - ssh-rsa AAAA... user@example.com

# Install the nginx web server and firewall packages
packages:
  - nginx
  - ufw

# Run commands after packages are installed to configure and start services
runcmd:
  - [ ufw, allow, 'WWW Full' ]
  - [ systemctl, enable, --now, nginx ]
```

The I/O Challenge & High-Performance Solution

A VM has no physical hardware. The hypervisor must provide virtual devices.

- **Emulation (Slow):** The hypervisor can pretend to be a real, common piece of hardware (like an Intel E1000 network card). This is compatible but slow.
- **Paravirtualization (Fast):** Modern systems use **VirtIO**. The guest OS has a special `virtio` driver that doesn't emulate real hardware but instead uses a highly efficient, standardized channel to communicate with the hypervisor for disk and network tasks.

Virtual Networking: NAT vs. Bridge Mode

- **NAT Mode (Default):** The VM shares your host's IP address. It's easy to set up and allows the guest to access the internet, but makes it difficult for other devices on your network to connect to the guest.
- **Bridge Mode:** The VM gets its own unique IP address on your local network, appearing as a separate physical device. This is ideal for running servers.

Device Access: USB & PCI Passthrough

You can grant a VM exclusive control over a physical device connected to your host.

- **USB Passthrough:** Gives a VM direct access to a USB device. This is essential for embedded development, allowing your Debian VM to directly program an **Arduino or ESP32** board.
- **PCI Passthrough:** Assigns a physical PCI device, like a powerful **GPU**, directly to a VM. This offers near-native performance for demanding tasks like gaming or machine learning.

How PCI Passthrough is Achieved

This advanced feature requires hardware support from the CPU and motherboard chipset, specifically the **IOMMU (Input-Output Memory Management Unit)**.

- **Intel's IOMMU:** VT-d
- **AMD's IOMMU:** AMD-Vi

The IOMMU creates a secure memory sandbox for the device, ensuring it can only access the memory of the VM it's assigned to. This prevents the device from interfering with the host OS or other VMs.

Introducing: Oracle VirtualBox

VirtualBox is a **Type-2 (hosted)** hypervisor that runs as a standard application on your existing OS. It is developed by Oracle and is free and open-source.

- **Who it's for:** Beginners, students, and desktop users who need an easy-to-use, graphical interface for running VMs.
- **Key Features:**
  - Cross-platform (Windows, macOS, Linux).
  - User-friendly graphical interface.
  - "Guest Additions" for seamless mouse integration, shared folders, and clipboard.
  - Easy-to-use snapshot functionality.

Installing VirtualBox

The process involves installing the main application and a separate Extension Pack for full functionality (like USB 2.0/3.0 support).

1. **Download:** Go to the official VirtualBox downloads page and download the package for your host OS (Windows, macOS, or Linux). Also download the **Extension Pack**.
2. **Install Application:** Run the installer for the main application.
3. **macOS Security:** On macOS, you must go to **System Settings > Privacy & Security** to **Allow** the system extension from Oracle.
4. **Install Extension Pack:** Double-click the downloaded `.vbox-extpack` file. VirtualBox will open and guide you through the installation.

Using VirtualBox

Creating and running a VM is a straightforward, wizard-driven process.

1. Click the **“New”** button to start the new VM wizard.
2. Assign a name, OS type, and RAM.
3. When prompted for a hard disk, you can create a new one or, for your class, choose **“Do not add a virtual hard disk”**.
4. Go to the VM’s **Settings > Storage** and click the “Add Hard Disk” icon to attach your provided `.vdi` file.
5. Review other settings like **Network** (`NAT` or `Bridged`) and **USB** (to enable passthrough).
6. Select the VM and click **“Start”**.

Introducing: QEMU + KVM

QEMU is a powerful machine emulator, and KVM (Kernel-based Virtual Machine) is the Linux kernel's built-in virtualization module. Together, they provide high-performance, **Type-1 (bare-metal)** virtualization on Linux.

- **Who it's for:** System administrators, developers, and power users who need flexibility, performance, and command-line control. It is the engine behind many large-scale cloud platforms.
- **Key Features:**
  - Extremely flexible and scriptable.
  - Can emulate a huge variety of CPU architectures (ARM, MIPS, etc.).
  - Near-native performance when used with KVM.
  - Advanced storage features with the `.qcow2` format

Installing QEMU + KVM

On Debian/Ubuntu-based systems, installation is done via the `apt` package manager.

1. **Install Packages:** Open a terminal and run the following command. `virt-manager` is a highly recommended graphical tool for managing QEMU/KVM VMs.

```
$ sudo apt update
$ sudo apt install qemu-system-x86 kvm virt-manager libvirt-daemon-system
```

2. **Add User to Groups:** Add your user to the `libvirt` and `kvm` groups to manage VMs without needing `sudo` for every action. You will need to log out and back in for this to take effect.

```
$ sudo adduser $USER libvirt
$ sudo adduser $USER kvm
```

Using QEMU + KVM

While `virt-manager` provides a GUI, using QEMU from the command line demonstrates its power.

1. **Create a Virtual Disk:** The `.qcow2` format is recommended. This command creates a 20GB disk that only grows as you add data.

```
$ qemu-img create -f qcow2 my_debian_disk.qcow2 20G
```

2. **Launch a VM:** This command starts a VM to install an OS from an ISO file.

```
$ qemu-system-x86_64 -enable-kvm -m 2048 -hda my_debian_disk.qcow2 \
-cdrom debian-13-netinst.iso -boot d
```

   - `-enable-kvm`: Use KVM for hardware acceleration (critical for performance).
   - `-m 2048`: Assign 2048 MB of RAM.
   - `-hda`: Specify the primary hard disk file.
   - `-cdrom`: Attach an ISO file as a virtual CD-ROM.
   - `-boot d`: Tell the VM to boot from the CD-ROM drive first.

Comparison: VirtualBox vs. QEMU/KVM

Feature	VirtualBox	QEMU + KVM
Type	Type-2 (Hosted)	Type-1 (Bare-Metal, via Linux Kernel)
Primary Platform	Cross-Platform (Windows, macOS, Linux)	Linux
Ease of Use	**Very High** (Graphical, wizard-driven)	**Medium to Low** (Command-line focused)
Performance	Good (Excellent for desktop use)	**Excellent** (Near-native speed)
Flexibility	Good (User-friendly options)	**Very High** (Highly configurable and scriptable)
Best For...	Students, desktop users, quick setups	Servers, developers, custom solutions, emulation

Support & Further Resources 📚

Bookmark these pages for quick reference.

- **Virtual Box:**
  - Manual
  - Networking
- **QEMU:**
  - Manual
  - Networking