Virtual Machines (VMs) vs Containers: What's the Difference and When to Use What?

If you've been hearing a lot about **containers** and **virtual machines (VMs)** in DevOps, cloud, or software development conversations — and you're wondering what the fuss is all about — you're not alone.

Both are technologies used to **run applications in isolated environments**, but they do it in **very different ways**. In this article, I'll break down the differences in **simple terms**, so you'll walk away knowing:

- What VMs and containers actually are
- How they're different
- When to use which one
- And why are containers becoming so popular

1. What Is a Virtual Machine (VM)?

A **Virtual Machine** is like a **computer inside your computer**.

When you create a VM, you're running an entire operating system (like Windows or Linux) on top of your existing system. It's made possible through something called a **hypervisor**, which sits between your hardware and your virtual OS.

Each VM contains:

- **A full operating system (OS)** (e.g., Windows, Linux).
- **Virtualized hardware** such as CPU, memory, and storage.
- **Your application stack.**

Example: Imagine running Windows 10 on your MacBook using VirtualBox or VMware. That's a virtual machine (VM).

Key Traits of VMs:

- Each VM has **its own full OS**, including kernel and libraries.
- It's **heavily isolated** from other VMs and from the host.
- It can be **slow to start** (boot time in minutes).
- **Takes more disk space and memory** (gigabytes).

2. What Is a Container?

A **container** is a lightweight package that includes your **application and everything it needs to run** (code, libraries, config), but it shares the same operating system **kernel** as the host.

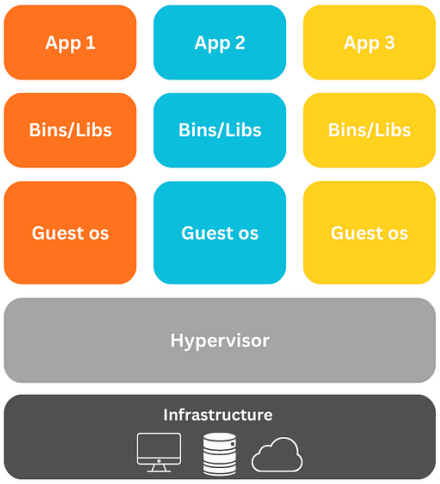
Unlike VMs, containers **don't run a full OS**. They use the host's OS kernel and are managed by container engines like **Docker**, **Podman**, or **containerd**.

Examples of containerization tools: Docker, Podman, Kubernetes (for container orchestration)

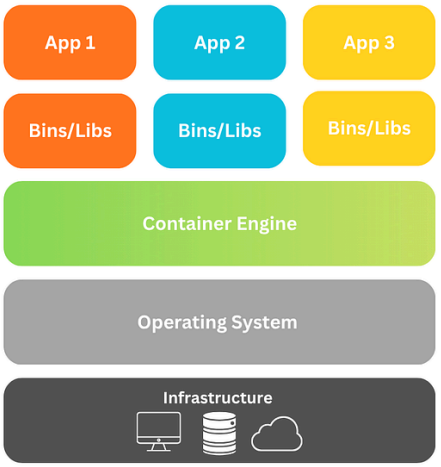
Example: Think of a container like a zipped folder with an app that runs instantly on any system with Docker installed.

Key Traits of Containers:

- They are **lightweight** and **start in seconds**.
- Use **less CPU and RAM** (megabytes).
- **Share the host OS kernel** (less isolation than VMs).
- Easily **portable across environments** (laptop, server, cloud).



VIRTUAL MACHINE(VM)



CONTAINERS

3. Key Differences Between Containers and VMs

Here's a detailed comparison of containers vs virtual machines, including their architecture and performance differences:

Zoom image will be displayed

Virtual Machine (VM)	Container
Runs a **full OS** (Guest OS) on top of the host OS using a hypervisor.	Shares the **host OS kernel**; does not require a separate OS for each application.
Heavy – each VM can consume **GBs** of storage due to the complete OS image.	Very lightweight – containers typically consume **MBs** of storage.
VMs take **minutes** to boot as they load an entire OS.	Containers start in **seconds** (no OS boot required).
Provides **full OS-level isolation** – each VM acts like a separate physical machine.	Provides **process-level isolation** – all containers share the same OS kernel.
Slower performance due to **hardware and OS virtualization layers** (hypervisor overhead).	Near-native performance as there's no OS virtualization overhead.
Less portable – moving VMs across environments is complex and requires compatible hypervisors.	Highly portable – container images can run anywhere with a container runtime (e.g., Docker, Podman).
Scaling is harder – the heavy nature of VMs limits how many can run on a host.	Easier to scale horizontally – you can run hundreds of containers on the same host.
VMs are more secure – **strong OS-level isolation** reduces the risk of breaches between environments.	Shared OS kernel makes containers **less secure by default**, but isolation can be enhanced with tools (e.g., gVisor, SELinux).
Best for **legacy apps, OS testing, running different OS types, and high-security workloads**.	Best for **microservices, CI/CD, cloud-native apps**, and **fast deployments**.
Managed with **hypervisors** (VMware, VirtualBox, KVM) and cloud providers (AWS EC2, Azure VMs).	Managed with **container engines** (Docker, Podman) and orchestrators (Kubernetes).
More expensive – consumes more CPU and memory resources per instance.	Cheaper to run – uses fewer resources, allowing more workloads per host.

4. When Should You Use What?

Use Virtual Machines when:

- You need **strong security isolation** (like in production servers).
- You want to run a **different OS** than the host (e.g., Windows on Linux).
- You're dealing with **legacy applications** or **monolithic systems**.

Use Containers when:

- You want to build and ship apps **quickly and consistently**.
- You're using **microservices** or modern **cloud-native** architecture.
- You need to **scale easily**, like in **Kubernetes** clusters.
- You care about **fast startup and lower resource usage**.

5. Why Are Containers Becoming So Popular?

Containers are rapidly becoming the **default choice** for modern application development. Here's why:

- **Portability Across Environments:** Container images can run anywhere with a container runtime.
- **Speed and Efficiency:** Containers start in **seconds** because they don't need a full OS boot.
- **Perfect for Microservices and DevOps:** Containers make **scaling and updating microservices easy.**

- **Better Resource Utilisation:** They share the host OS kernel, allowing **more apps per host.**
- **Seamless CI/CD Integration:** Containers are ideal for **continuous integration and deployment pipelines.**
- **Cloud-Native Ready:** Tools like **Kubernetes** and platforms like AWS and Azure fully support containers.
- Strong Ecosystem and Community: **Docker, Podman, containerd, Kubernetes**

Final Thoughts

If you're just starting out, here's the **one-line summary**:

VMs are like houses with their own land and utilities — fully independent but heavier to build and maintain. Containers are like apartments in the same building — lighter, faster, and easier to manage. In fact, many companies use **both together** — running containers on top of VMs for maximum flexibility and security.

Both are powerful tools. But for modern cloud-native development, **containers are leading the way** due to their speed, portability, and scalability.

Which do you prefer — Containers or VMs? Share your thoughts in the comments below!

If you found this article helpful, give it a clap on Medium and share it with your DevOps friends!

Bonus: Want to Try It Out?

If you want a hands-on experience:

- Install Docker
- Try running your first container:

```
docker run hello-world
```

Or spin up a VM using VirtualBox and compare the experience!