Modern software needs to run reliably across different machines and environments. However, differences in hardware and operating systems make this challenging. To solve this, technologies like **Virtual Machines** and **Docker containers** were developed. This article explains these concepts step by step, starting from bare metal systems.

---

1. Bare Metal Systems

A **bare metal system** refers to a computer running directly on physical hardware without any virtualization layer.

Core Hardware Components

- **CPU (Central Processing Unit)**
  Executes program instructions and performs computations.

- **RAM (Random Access Memory)**
  Stores temporary working data used by running programs.

- **Disk (SSD/HDD)**
  Provides long-term persistent storage for files and applications.

- **ROM (Read-Only Memory)**
  Stores firmware instructions (BIOS/UEFI) used during system boot, not application data.

Role of the Operating System

The **Operating System (OS)**:

- Manages hardware resources

- Allocates CPU time and memory to applications

- Handles file systems, networking, and security

Limitation of Bare Metal Systems

Because operating systems differ across machines (Linux, macOS, Windows), applications often depend on OS-specific libraries and configurations. This makes it difficult to package software that runs consistently everywhere, limiting **portability and scalability**.

---

2. Virtual Machines (VMs)

Virtual machines address portability issues by **virtualizing hardware**.

How Virtual Machines Work

A virtual machine runs a complete **guest operating system** on top of physical hardware using a **hypervisor**.

Key Components

- **Host Machine**
  The physical server providing CPU, memory, and disk.

- **Hypervisor**
  A layer that manages and allocates hardware resources to virtual machines.

- **Guest OS**
  A full operating system running inside each VM.

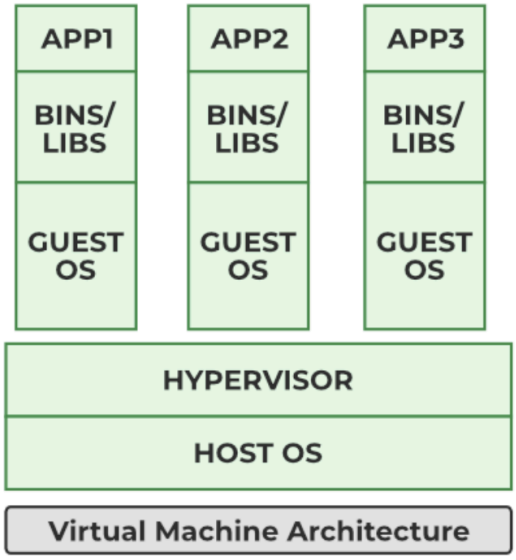
Multiple virtual machines can run concurrently on the same physical hardware, each believing it owns the entire system.

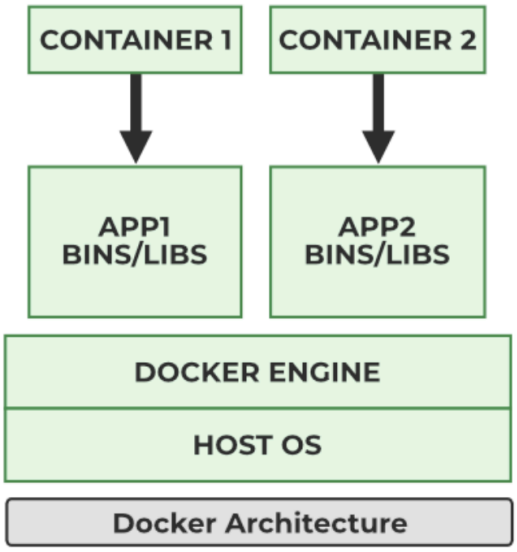
Advantages of Virtual Machines

- Strong isolation between applications
- Each VM has its own kernel and OS
- Applications can be packaged with their OS and moved across systems

Limitations of Virtual Machines

- Heavy memory and storage usage
- Slower startup times
- Higher CPU overhead due to full OS virtualization





3. Docker and Containers

Docker offers a more lightweight alternative to virtual machines through **containerization**.

What Docker Is

Docker is an open-source platform that enables developers to **build, package, deploy, and run applications** using containers.

How Containers Differ from Virtual Machines

Containers **do not virtualize hardware** and **do not include a guest operating system**. Instead, they:

- Share the **host operating system's kernel**
- Package only the application and its user-space dependencies
- Use OS-level isolation mechanisms

A Critical Assumption Docker Makes

Docker containers **assume that a Linux kernel is available at runtime**.

This is a fundamental design principle of Docker:

- Docker images **do not include a kernel**
- Docker containers depend on **Linux system calls**
- Docker uses Linux kernel features such as:
    - namespaces (for isolation)
    - cgroups (for resource control)

Because of this, Docker containers can only run when a compatible **Linux kernel** is present.

This makes containers significantly more efficient than virtual machines.

---

4. How Docker Runs on Different Operating Systems (Expanded)

Docker on Linux

On a Linux system, Docker containers run **natively**.

```
Linux Host
└── Linux Kernel
    └── Docker Engine
        └── Containers
```

No virtual machine is required because the Linux kernel is already available.

Docker on macOS and Windows

macOS and Windows do **not** provide a Linux kernel. To satisfy Docker's runtime requirement, Docker Desktop automatically creates and manages a **lightweight Linux virtual machine**.

```
macOS / Windows
└── Docker Desktop
    └── Linux Virtual Machine
        └── Linux Kernel
            └── Docker Containers
```

In this setup:

- Docker still runs **Linux containers**
- The host OS (macOS or Windows) does not run containers directly
- The Linux VM exists solely to provide the required kernel

This design allows the **same Docker image** to run consistently across different host operating systems.

---

5. Why This Design Matters

Because Docker assumes the presence of a Linux kernel:

- Docker images are **highly portable**
- The same image can be built once and run anywhere Linux is available
- Containers remain lightweight and fast

However, this also means:

- Docker cannot run Linux containers directly on non-Linux kernels
- A virtualization layer is unavoidable on macOS and Windows

4. Docker Architecture

Dockerfile

A **Dockerfile** is a text file containing instructions that describe:

- The base environment
- Dependencies to install
- How to build and run the application

It acts as a **blueprint** for creating Docker images.

Docker Image

A **Docker image** is a read-only template created from a Dockerfile.
It contains:

- Application code
- Runtime dependencies
- System libraries (but not a kernel)

Docker Container

A **container** is a running instance of a Docker image.
It is:

- Lightweight
- Isolated
- Ephemeral by default (data is not persisted unless volumes are used)

Containers are essentially **isolated processes** running on the host system.

- **Docker Engine:** Docker Engine has a core part docker daemon , that handles the creation and management of containers.
- **Docker Hub:** It is a cloud based repository that is used for finding and sharing the container images

---

5. Docker Across Different Operating Systems

- **Linux**: Containers run directly on the Linux kernel.
- **macOS and Windows**: Docker Desktop runs a lightweight Linux virtual machine to provide the required kernel.

This design allows the **same Docker image** to run consistently across different operating systems.

---

6. Why Docker Is Faster and More Efficient

Compared to virtual machines, Docker:

- Starts in milliseconds instead of minutes
- Uses significantly less memory
- Allows higher application density per machine
- Simplifies deployment and scaling

However, because containers share the host kernel, they provide slightly weaker isolation than full virtual machines.

Further Readings:

https://www.geeksforgeeks.org/devops/introduction-to-docker/
https://www.ibm.com/think/topics/docker