International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2019): 7.583

Virtualization vs Containerization: Differences and Use Cases

Pallavi Priya Patharlagadda

Engineering, United States of America

Abstract: ***Virtualization and Containerization are two most frequently used mechanisms to deploy multiple, isolated services on a single platform. But the way both technologies function is different. Let's understand both the concepts and then derive the use cases where Virtualization is preferred and where Containerization is preferred.***

Keywords: virtualization, containerization, deployment, isolated services, technology comparison

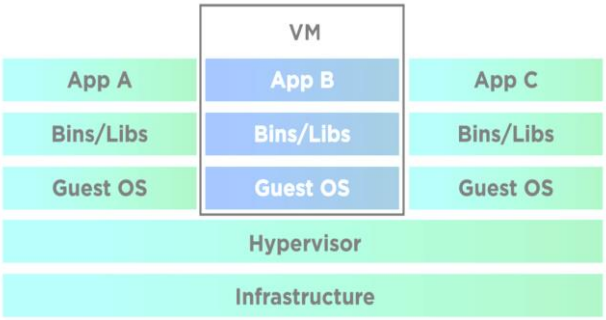
1. Problem Statement

As the world advances, the requirement to improve the scalability and performance has increased. At the same time, companies are looking for cost reduction and standardize software deployments across multiple machines and platforms. Virtualization and Containerization are the two approaches that can be followed to make efficient use of the resources. Since, Containerization is also a kind of virtualization, often people get confused between both. This paper speaks in detail about both the topics, their advantages and disadvantages and use cases on when virtualization or containerization should be preferred over another.

2. Introduction

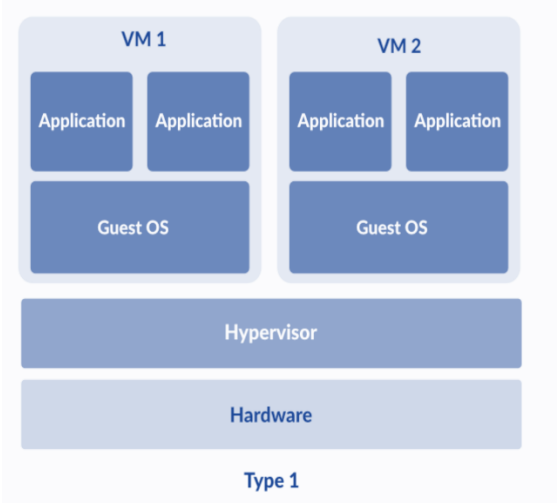
Virtualization

Virtualization is the technique of using software to build a layer of abstraction over hardware that allows a single computer's hardware to be split into several virtual computers. Each of those virtual computers (known as "guests") uses part of the hardware resources of the main computer (known as a "host").



a) This is accomplished using hypervisor software. Virtual machine monitors (VMMs), commonly referred to as hypervisors, allow numerous guest operating systems to run on top of a host operating system and share the same physical computing resources under the control of the host operating system. This essentially enables the operating system and programs to be abstracted from the hardware on the actual computer. There are two main groups into which hypervisors fall. Two types of hypervisors: Type 1 and Type 2.

b) Type 1 Hypervisor: Operating directly on the physical hardware of the underlying machine, a Type 1 hypervisor interacts with its CPU, memory, and physical storage. Because of this, Type 1 hypervisors are sometimes called bare - metal hypervisors. The host operating system is replaced with a Type 1 hypervisor.



Advantages of Type - 1 hypervisor

- High performance: Since they are not restricted by the built - in OS constraints, these offer excellent performance.
- Extremely secure: They are shielded from OS defects and vulnerabilities since they operate directly on the physical hardware without the need for an underlying operating system. This guarantees that all virtual machines (VMs) are segregated from any malicious malware.

Disadvantages of Type - 1 Hypervisor:

- Complexity: Type 1 hypervisor management and setup are challenging since they necessitate an understanding of hardware settings.
- Hardware compatibility: Every hypervisor has a list of compatible hardware on it. This results in an extra cost when using an alternative hypervisor.
- Use case for using Type 1 Hypervisor: Web servers, data centers, enterprise computing workload scenarios, and other applications with a fixed user base are common uses for type 1 hypervisors. To provide the most performant virtual machines (VMs) for the underlying physical hardware, cloud computing environments use bare metal hypervisors. Additionally, cloud providers offer virtual machines (VMs) as cloud instances that you can access via

APIs, abstracting away type 1 hypervisor management. Examples of Type - 1 Hypervisors VMware vSphere with ESX/ESXi, Microsoft Hyper - V, Citrix Hypervisor (Xen Server).

- Type 2 Hypervisor: A Type 2 hypervisor doesn't run directly on the underlying hardware. Instead, interacts with the underlying host machine hardware through the host machine's operating system. It interacts with the operating system to obtain underlying system resources. However, the host operating system prioritizes its own functions and applications over the virtual workload.

Advantages of Type - 2 hypervisor:

Ease of use: These are easy to setup.
No Hardware compatibility issues: Since the OS takes care of the underlying hardware, Developer need not worry on the compatibility issues.

Disadvantages:

1) Low Performance: Since Type 2 hypervisor access computing, memory, and network resources via host OS, it introduces latency issues that can affect performance.
2) Security Risks: It introduces potential security risks if an attacker compromises the host OS as they could then manipulate any guest OS running in the Type 2 hypervisor.

Use case for Type2 Hypervisors:
Type 2 hypervisors are most often used in desktop and development environments, where workloads are not as resource - intensive or critical to operations. They're also preferred in cases where users want to simultaneously use two or more operating systems but only have access to one machine. Examples of Type 2 Hypervisors – VMware Workstation Pro/VMware Fusion, Oracle VirtualBox, etc.

	Bare Metal hypervisor	Hosted hypervisor
Also known as	Type 1 hypervisor.	Type 2 hypervisor.
Runs on	Underlying physical host machine hardware.	Underlying operating system (host OS).
Best suites for	Large, resource-intensive, or fixed-use workloads.	Desktop and development environments.
Negotiation on dedicated resources	Yes.	No.
Knowledge	System administrator-level knowledge.	Basic user knowledge.
Examples	VMware ESXi, Microsoft Hyper-V, KVM.	Oracle VM VirtualBox, VMware Workstation, Microsoft Virtual PC.

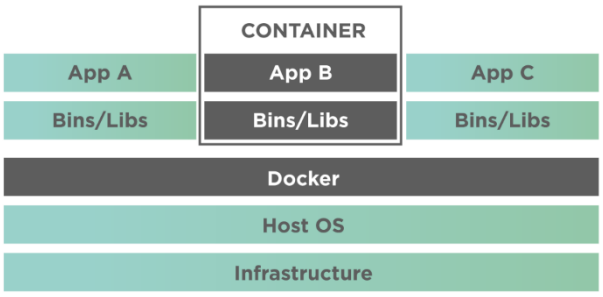
3. Containerization

Containerization is a form of virtualization that bundles an application with its code, libraries, dependencies, and everything else needed for it to run inside a component known as a container. Containers are lightweight and portable, running consistently across any computing platform. containers share the host system's user space, while still maintaining its individual system processes, environment variables, and libraries. The isolation among containers is achieved using Linux namespaces and cgroups.

Process Containers was launched by Google in 2006 and was designed for limiting, accounting and isolating resource usage like CPU, memory, disk I/O, network etc. for a collection of processes. It was renamed as Control Groups (cgroups) and merged to Linux kernel 2.6.24.

LXC (Linux Containers) was the first and the most complete implementation of Linux container manager. It was implemented in 2008 using cgroups and Linux namespaces, and it works on a single Linux kernel without requiring any patches.

Docker emerged in 2013 and containers exploded in popularity. The growth of Docker and container use goes together. Docker used LXC in its initial stages and later replaced that container manager with its own library, libcontainer. Docker separated itself from the LXC by offering an entire ecosystem for container management.
Docker provides tooling and a platform to manage the lifecycle of your containers.



Numerous advantages come with containerization, such as scalability, portability, and quick deployment. It lowers the chance of system instability or application conflicts by enabling developers to create predictable environments that are segregated from other applications. They can also package their software along with all its dependencies so that it can run on any system that has a container engine installed, no matter what configuration it has.

Moreover, the microservices architecture—which divides applications into discrete, standalone services—is supported by containerization. This method enables more efficient administration of complicated applications as well as quicker and more dependable deployment.

4. Containers in Linux: Namespaces and C Groups

In this section, we delve into the Linux specifics of what we use to implement containers. In essence, though, they are extensions of existing APIs: C Groups are essentially an extension of Resource Limits (POSIX RLIMITs) applied to groups of processes instead of to single processes. Namespaces are likewise sophisticated extensions of the chroot () separation system applied to a set of different subsystems. The object of this section is to explain the principles of operation rather than give practical examples (which would be a whole article in its own right. The information in this section, being very Linux specific, may have changed since then

1) CGroups:

CGroups can be thought of as resource controllers (or limiters) on types of resources. The thing about most CGroups is that the control applies to a group of processes (hence the interior of the container becomes the group) that it's inherited across forks, and the CGroups can be set up hierarchically. The current CGroups are:

- blkio—controls block devices
- cpu and cpuacct—controls CPU resources
- cpuset—controls CPU affinity for a group of processes.
- devices—controls device visibility, effectively by gating the mknod () and open () calls within the container
- freezer—allows arbitrary suspend and resume of groups of processes
- hugetlb—controls access to huge pages, something very Linux specific
- memory—currently controls user memory allocation but soon will control both user and kernel memory allocations
- net_cls and net_prio—controls packet classification and prioritization
- perf_event—controls access to performance events as you can see from the brief descriptions, they're much more extensive than the old RLIMIT controls. With all these controllers, you can effectively isolate one container from another in such a way that whatever the group of processes within the container do, they cannot have any external influence on a different container (provided they've been configured not to, of course).

2) Namespaces

Although, simplistically, we've described namespaces as being huge extensions of chroot (), in practice, they're much more subtle and sophisticated. In Linux there are six namespaces:

- Network—tags a network interface
- PID—does a subtree from the fork, remapping the visible PID to 1 so that init can work.
- UTS—allows specifying new host and NIS names in the kernel
- IPC—separates the system V IPC namespace on a per - container basis
- Mount—allows each container to have a separate file - system root
- User—does a prescribed remapping between UIDs in the host and container.

The namespace separation is applied as part of the clone () flags and is inherited across forks. The big difference from chroot () is that namespaces tag resources and any tagged resources may disappear from the parent namespace altogether (although some namespaces, like PID and user are simply remapping of resources in the parent namespace). Container security guarantees are provided by the user namespace, which maps UID 0 within the container (the root user and up, including well known UIDs like bin) to unused UIDs in the host, meaning that if the apparent root user in the container ever breaks out of the container, it is completely unprivileged in the host.

5. Differences between virtualization and containerization

a) Resource Overhead

From a resource overhead perspective, containerization clearly outperforms virtualization. Containers are much lighter and use less resources because they share the operating system of the host system rather than requiring the use of a separate operating system. However, since each virtual machine needs its own operating system, there is an increased overhead, particularly when multiple VMs are operating on the same host system.

b) Startup Time

Because they don't need to start an entire operating system, containers typically start up faster than virtual machines (VMs). Booting up virtual machines takes a lot longer. Because of their increased flexibility and ability to be stopped and restarted as needed, containers support immutability—the property that a resource maintains its original state even after it has been deployed.

c) Portability

Virtual machines and containers both provide a high level of portability. Because they bundle the application and all of its dependencies into a single unit that can run on any system that supports the container platform, containers do have a slight advantage over other options. Even though they are portable, virtual machines rely more on the underlying hardware.

d) Security Isolation

Virtual machines are better in terms of security isolation. A security breach in one virtual machine (VM) usually has no effect on the others because each VM is isolated from the host system and other VMs (though it is possible to compromise the hypervisor and take control of all VMs on the device). Although isolated from one another, containers share the operating system of the host system, so a breach in one container may potentially affect other containers.

e) Scalability and Management

Due to their lightweight design and quick startup time, containers are the best option for efficiently and quickly

scaling applications. Additionally, they work well with the microservices architecture, which makes managing complicated applications easier. Although scalable as well, virtual machines are less appropriate for microservices and distributed applications because they require more resources and take longer to start up.

f) Multi Operating Systems on a Single Box

Since each virtual machine installs its own operating system, virtualization enables the use of multiple operating systems on a single operating system. Thus, we can install both Linux and Windows virtual machines on a single kernel. However, in the event of containerization, the host kernel is shared. As a result, you cannot run two separate operating systems—such as Windows running on a Linux kernel—on the same physical machine.

Below is the comparison between virtualization and Containerization.

Area	Virtualization	Containerization
Isolation	Provides complete isolation from the host operating system and the other VMs.	Typically provides lightweight isolation from the host and other containers, but doesn't provide as strong a security boundary as a VM
Operating System	Runs a complete operating system including the kernel, thus requiring more system resources such as CPU, memory, and storage	Runs the user-mode portion of an operating system, and can be tailored to contain just the needed services for your app using fewer system resources
Guest compatibility	Runs just about any operating system inside the virtual machine	Runs on the same operating system version as the host
Deployment	Deploy individual VMs by using Hypervisor software	Deploy individual containers by using Docker or deploy multiple containers by using an orchestrator such
Persistent storage	Use a Virtual Hard Disk (VHD) for local storage for a single VM or a Server Message Block (SMB) file share for storage shared by multiple servers	Use local disks for local storage for a single node or SMB for storage shared by multiple nodes or servers
Load balancing	Virtual machine load balancing is done by running VMs in other servers in a failover cluster	An orchestrator can automatically start or stop containers on cluster nodes to manage changes in load and availability.
Networking	Uses virtual network adapters	Uses an isolated view of a virtual network adapter. Thus, provides a little less virtualization

6. Use Cases for Virtualization:

a) Legacy Applications

Legacy applications are frequently viewed as a burden in the software industry. These are programs that are usually hard to update or maintain because they were created with outdated technology. Nevertheless, they are frequently indispensable to the functioning of businesses and cannot be thrown away. This is the sweet spot for virtualization.

These legacy apps can run on their original operating systems thanks to virtualization, even if the underlying hardware has been updated. This implies that companies won't need to invest in costly or time - consuming updates in order to keep using these apps. Additionally, the sandboxed environment that virtualization creates shields the rest of the system from any security flaws in these older apps.

b) High Isolation Environments

Settings Requiring a High Degree of Isolation Strong application isolation is a critical requirement in environments where virtualization performs exceptionally well. This is especially helpful in high - security settings where an application breach shouldn't have an impact on other applications.

To separate applications belonging to different users from one another, for example, virtualization can be utilized in a data center that serves several organizations. The attacker would not be able to access applications running on other virtual machines, even if one user's application was compromised.

c) IaaS Scenarios

An example of a cloud computing model is Infrastructure as a Service (IaaS), which offers networks, storage, and virtual computers as services. IaaS is based on virtualization as a key technology. By operating several virtual machines (VMs) on the same physical hardware, it enables cloud providers to make effective use of their hardware resources.

Furthermore, resource scalability based on demand is made possible by virtualization. It is simple to assign additional virtual machines to a client in case they require more resources.

7. Use Cases for Containerization

Here are the main use cases for containerized applications:

a) Microservices Architectures

Microservices architectures are one of the primary use cases for containerization. An application is divided into tiny, autonomous services that interact with one another in a microservices architecture. There are numerous advantages to this strategy, such as increased scalability and simpler maintenance.

Microservices are ideally suited for containers. They give every service a uniform environment, guaranteeing that it operates reliably on various platforms. Furthermore, services or service instances cannot clash with one another because containers are isolated from one another.

b) CI/CD

Software developers use continuous integration/continuous deployment (CI/CD) to integrate their code into a shared repository frequently—typically multiple times per day. After that, every integration is deployed and tested automatically. Teams can identify and address issues earlier with this method, which produces software that is of a higher caliber.

One important component of CI/CD is containerization.

Testing can be done in a consistent environment with containers, which guarantees repeatability and dependability. Additionally, containers are simple to deploy to production, which speeds up and improves the efficiency of the deployment process.

c) PaaS Scenarios

Under the Platform as a Service (PaaS) cloud computing model, developers can build, test, and deploy apps on a platform that is provided by the provider. An operating system, middleware, and runtime environment are often included in this platform.

PaaS requires containerization as a necessary component. By allowing several containers to run on a single host, it enables providers to make effective use of their resources. It also gives developers a standardized environment, which facilitates the development and deployment of applications.

8. Use Virtualization and Containerization Together

I'm curious as to why someone would want to combine virtual machines and containers. Well, binaries and libraries are shared by containers with the host's OS kernel. Since most Linux distributions are based on the same kernel, running Linux containers across distributions is not a problem.

For instance, CentOS - based hosts can run Ubuntu containers effectively. It is not possible to run Windows containers on Linux hosts or vice versa due to the kernel sharing fact. Create a virtual machine (VM) on the appropriate host in order to use these containers. To run Windows containers, for instance, you can set up a Windows virtual machine on a Linux host. Because a virtual machine runs on its own operating system, the operating system can support the container engine, making this possible.

By isolating a container within a virtual machine, vulnerability attacks are restricted in their reach. On a bare - metal server, for instance, if 500 containers share an OS kernel and the operating system malfunctions, all 500 containers become compromised. On the other hand, only those containers are impacted if a virtual machine (VM) hosting 50 or fewer containers is compromised. Other virtual machines running distinct containers on the same server or cluster are unaffected by this failure.

To accomplish capacity optimization, containers and virtual machines can be integrated. Because virtualization makes server utilization easier, it is widely used in the enterprise IT sector. Multiple virtual machines (VMs) can be hosted on a single server, and each VM can host multiple container hosts. Additionally, several conventional monolithic virtual machines (VMs) can be hosted on a single server. IT managers can optimize the use of the physical server by integrating containers with conventional monolithic virtual machines.

9. Conclusion

In this paper, we have examined how virtualization and containerization are similar in that they offer isolation. To put it briefly, containers are inexpensive to use because they perform far less. In addition to offering fully virtualized hardware layers, virtual machines are capable of much more, but at a high cost.

Containerization, which entails enclosing an application with its operating system in a container, can be thought of as a lightweight substitute for full machine virtualization. Whereas the other does not offer a useful application, each has advantages and particular situations. A few examples of how each can be used are discussed. The user can select between virtualization and containerization based on which system suits them the best in the given situation.

References

[1] https: //www.toolsqa. com/docker/understanding - containerization - and - virtualization/
[2] https: //www.aquasec. com/blog/a - brief - history - of - containers - from - 1970s - chroot - to - docker - 2016/