Virtual machines vs. containers

Johan Luttu

April 2019

1 Introduction

Both virtual machines and containers can be used to virtualize isolated environments where software can be deployed. They're similar in many ways, which can explain that there's a common misconception that containers serve the same purpose as virtual machines but in a new, better way [1][2]. However, this isn't necessarily true. Virtual machines and containers have some major differences which make them more or less suitable for different problems. This essay will discuss how they differ and when to use one or/and the other.

2 Background

In order to get a deeper understanding of the subject, it's required to have knowledge about operating systems. This is due to the fact that concepts which have to do with how operating systems work will turn out to be the key differences between virtual machines and containers. In order for this essay to be self-contained, concepts related to operating systems will be explained. If you're already deeply familiar with topics such as "what is a kernel?" you might want to skip this section.

2.1 Operating system kernel

The kernel is the core part of an operating system. It controls everything in the system. It decides which programs get to be executed on the CPU by scheduling processes, and it's responsible for allocating memory to the processes. It also communicates with I/O devices such as a hard drive or a network adapter, etc. [3].

2.2 Linux cgroups (control groups)

Cgroups is a feature of the Linux kernel that can be used to control groups of processes, hence the name control groups This feature allows you to group a set of processes that run as a related unit, and that group of running processes can be controlled with respect to how much resources of the host they're allowed to consume. Resources in terms of memory [5][6], CPU utilization [7], how much I/O they do (I/O both over network and to the hard drive and other devices) [8]. Accounting is also provided as a part of the feature, which consists of measuring the usage of resources by a particular group. Cgroups can also be used to freeze groups of processes and save snapshots of applications' states (so they can be restored after an eventual failure) [9]. Another thing about cgroups that is important to know in the context of this essay is that cgroups may be nested, which means that a cgroup can be a parent of another cgroup.

2.3 Linux kernel namespaces

A namespace is another kernel feature that can restrict the view of the system. Instead of showing you every aspect of a running system, it shows you a more narrow perception of the system. This way you get the illusion that you're running on a system that has fewer interfaces. For instance, if you're starting up a Linux box and logging in for the first time, you're in the root. You're going to see all running processes, the full view of all the file systems. Every network interface, every bridge, every tunnel interface, everything is going to be visible to you. However, if you enter a namespace, that view can be restricted [10].

2.4 Hypervisors

A hypervisor, also called a virtual machine monitor can be software, firmware or even hardware that creates and runs virtual machines. Where it runs depends on what type of hypervisor it is. There are two different types. One type is "Type-1", bare-metal hypervisors, also called native hypervisors. They run directly on the hardware of the host machine and manage guest operating systems. The other type is called "Type-2" or hosted hypervisors, which don't run on hardware but on software, on top of operating systems. [11]

3 Virtual machines vs. containers

This section is going to be straight forward. First, containers and virtual machines are going to be explained. Then, the actual difference between them is going to be discussed, with regards to runtime performance and security. I've decided to not compare the efficiency when it comes to deploying and scaling services with both technologies because I wanted to put more focus on the on performance and security since it interested me more.

3.1 Containers

The word container can be used to cover multiple things, which can be confusing to some. At its most basic, the notion and the concept of a container at runtime are that it's basically a sandbox for a process where resources and views of the system are restricted. To be more detailed, what all things that could be called containers have in common is that they're depending on Linux cgroups, Linux namespaces, a container image, and a related life cycle to function. By using namespaces, a container can get an isolated workspace where access to processes, network interfaces, IPC resources, the filesystem, and the kernel is restricted. The use of cgroups allows a container to be allocated hardware resources such as CPU utilization and memory. Containers share the kernel of the host operating system. [17]

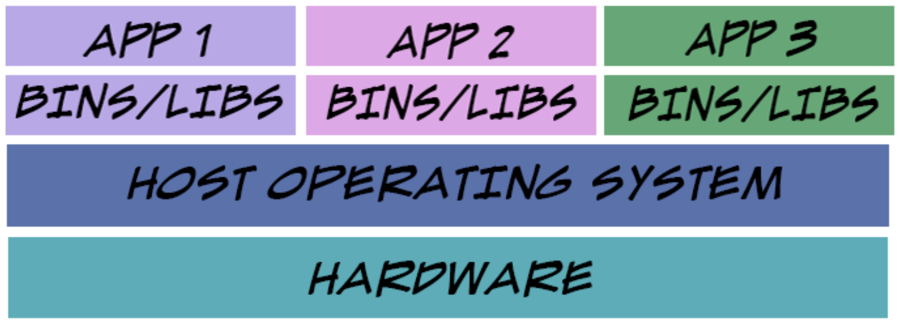

Figure 1: An overview of containers

As security measures, containers make use of other kernel features. All features won't be touched in this essay. For instance, the docker engine uses AppArmor and Seccomp which restrict programs' capabilities by for instance filtering system calls issued by programs and performing permission checks [4].

3.1.1 Container image

While cgroups and namespaces make up a container representation at runtime, a container image is basically a binary representation of a container. A docker image can be created by using a dockerfile. Each line of instructions in a dockerfile will create a new layer for the image. [14] There's a notion of a parent-child relationship implemented for container images, which allows images to easily be extended or be built on other container images. This is also beneficial when you want the image uploaded to a version control system, then you can just push or pull the latest layers. The container images also allow smooth portability, services can easily be deployed if an image contains all the required dependencies.

3.2 Virtual machines

Virtual machines are emulated isolated systems. Each virtual machine gets their own kernel, and they believe that they're running directly on the hardware. A hypervisor, of type1 or type2, is used to map the virtual machines' operations to the underlying hardware or operating system. The fact that the hypervisor interferes can lead to poor performance on weaker machines, and it especially starts to shine through if you try to run virtual machines within virtual machines - if you try to nest them. [16]

Virtual machines must host their own operating system in their hypervisor environment, referred to as the guest operating system. This includes everything that comes with the operating system: libraries, binaries, and applications, etc. This leads to a lot of overhead as it consumes a lot of system resources. Especially when virtual machines are running on the same host machine as they all have their own guest OS and pre-allocate resources. This makes them heavier than containers in the sense that they pre-allocate memory and that they emulate a whole system instead of just being an isolated process. This is why containers start faster. [16]

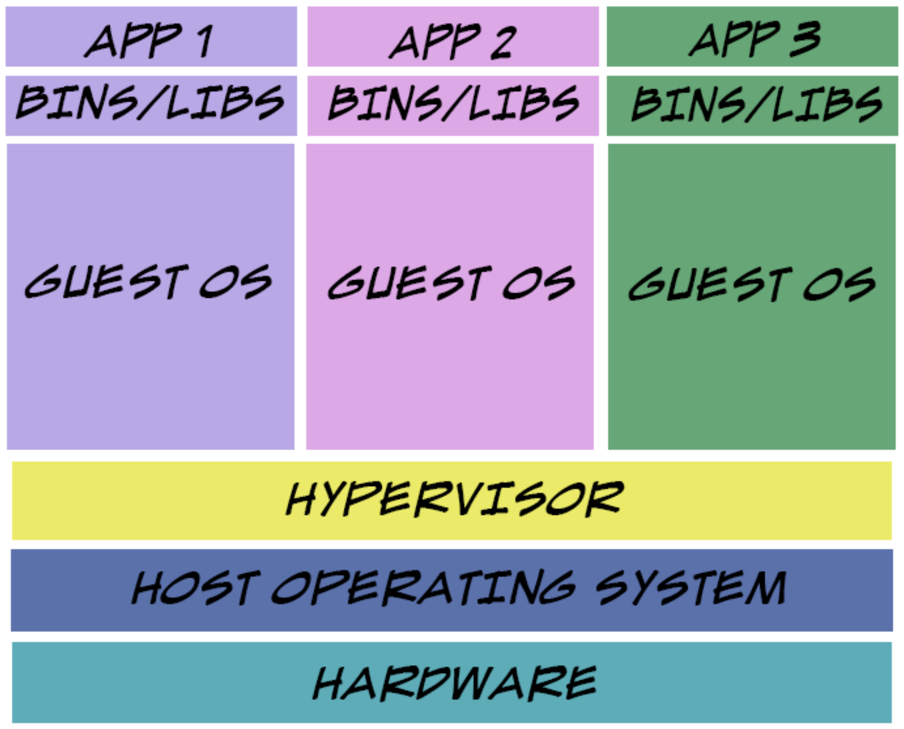

Figure 2: An overview of a system with virtual machines.

3.3 The difference

3.3.1 Performance

3.3.1.1 Runtime performance

The performance when it comes to processes running in a container is very similar to the native performance (when a process would be running on bare metal). [17]. This has to do with that a container is basically just an isolated process, that's been achieved with namespaces and cgroups. The cgroups and namespaces only control the teardown and the startup of the process, which by the way is much faster with containers compared to virtual machines. So, given that the containers share the kernel with the host, as said before, it doesn't change the execution path, even if a container's communication with the kernel can be restricted [10]. Containerized processes communicate directly with the kernel. This is why I've chosen to omit the container engine layer, above the host operating system layer in Figure 1, which some people like to display when they're illustrating a container overview like in the figure.

However, when it comes to virtual machines it's a different story. When processes run in a VM there's always a Type 1 or Type 2 hypervisor that interferes. The hypervisor must map the operations from within the guest operating system's own kernel to the underlying system that the hypervisor is running on.

3.3.1.2 Resources

One thing that makes containers differ from virtual machines is that they don't pre-allocate resources. For instance, it's possible to specify how much memory a container needs, but it won't pre-allocate it. This can lead to more efficient use of resources where more work is happening on the same host.

3.3.2 Security

A common threat when it comes to containers or virtual machines is an escape exploit. When malicious software breaks out of the sandbox environment of the container or the virtual machine, it can be phrased as it has "escaped". So, how secure are containers and virtual machines from neighboring containers or virtual machines on the same host? Between two neighboring virtual machines, the attack surface isn't that wide [15]. The escape has to happen through the hypervisor [15]. On the other hand, when it comes to containers, the risk profile is more complicated. Since containers share the same kernel of the host operating system, there are numerous ways of an escape to happen. This is due to the fact that there are over 300 different syscalls, at least for Linux 5.0 [12]. This makes it much more difficult to defend against.

Knowing that the attack surface between neighbouring containers on the same host is bigger than between neighboring virtual machines on the same host, one can say that the "barrier" is stronger between neighboring virtual machines on the same host. However, this doesn't mean that it isn't possible to make containers as secure as virtual machines. There exist techniques to limit the attack surface between neighboring containers. One example is to set a policy that enforces that the containers are by default not allowed to do anything with the kernel except the things that are explicitly allowed [13].

3.4 When to use what?

The use cases of these technologies aren't set in stone. When you get to choose between them it shouldn't be an exclusive choice of only using one of the technologies. It's possible to combine the use of virtual machines and containers by for instance putting a container within a virtual machine. For instance, if you care about that stronger barrier between neighboring virtual machines and decide to use a one, it can still be beneficial to use a container within it if you care about the portability which the container image allows, etc.

In many cases, it's probably better to use a container over a virtual machine, but a concrete use case for when you would want to use a virtual machine over a container could be if you're running a web hotel. Then it would probably be a good idea to give each customer an own virtual machine for security reasons, given the stronger isolation. It also guarantees the customer that the promised resources that have been paid for are available.

A use case for when you would want to use containers over virtual machines could be when you want to deploy multiple instances of a single application or service. By using containers, this can be done more efficiently than with virtual machines since the containers don't consume as many resources, which allows full utilization of the hardware of the machine that you're deploying your apps or services on.

4 Conclusion

Briefly speaking, containers are in runtime isolated processes with restricted resources (they do however not pre-allocate memory resources) that share the kernel of the host operating system. They're portable as you can pack an environment with the required dependencies and binaries for an application in a container image that can be layered. You can also pack dependencies and applications within a VM image, but a whole guest operating system will be packed in it, it's not as light as a container image.

Virtual machines are emulated systems with own kernels and stronger isolation between neighboring virtual environments. They host their own guest OS in their hypervisor environment, which makes them heavy.

When you execute processes in containers it's like running them on bare metal. In virtual machines, a hypervisor interferes by mapping the operations to the underlying operating system or hardware. The startup and teardown speed is also much faster for containers.

References

[1] https://stackoverflow.com/questions/16647069/should-i-use-vagrant-or-docker-for-creating-an-isolated-environment/21314566#21314566 - accessed 2019-04-29

[2] https://stackoverflow.com/questions/16047306/how-is-docker-different-from-a-virtual-machine accessed 2019-04-30

[3] "Kernel". Linfo. Bellevue Linux Users Group.

[4] https://docs.docker.com/engine/security/ - accessed 2019-04-30

[5] Corbet, J. (31 July 2007). "Controlling memory use in containers". LWN.

[6] Singh, B., Srinivasan, V. (July 2007). "Containers: Challenges with the memory resource controller and its performance". Ottawa Linux Symposium.

[7] Corbet, J. (23 October 2007). "Kernel space: Fair user scheduling for Linux". Network World.

[8] Kamkamezawa Hiroyu (19 November 2008). Cgroup and Memory Resource Controller. Japan Linux Symposium.

[9] Hansen, D. Resource Management. Linux Foundation.

[10] http://man7.org/linux/man-pages/man7/namespaces.7.html - accessed 2019-04-30

[11] Meier, Shannon (2008). "IBM Systems Virtualization: Servers, Storage, and Software" p. 2, 15, 20.

[12] https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_64.tbl - accessed 2019-04-30

[13] Linux kernel capabilities. Docker. - accessed 2019-04-30

[14] https://docs.docker.com/engine/reference/commandline/images/. Docker. - accessed 2019-04-30

[15] Sabahi, F., Member, IEEE (2012, February). "Secure Virtualization for Cloud Environment Using Hypervisor-based Technology" p. 45, Singapore. - accessed 2019-04-30

[16] Barrett, D., Kipper, G. (2010). Virtualization and forensics a digital forensic investigator's guide to virtual environments. p. 8, 13. Amsterdam ; Boston: Elsevier / Syngress.

[17] Felter, W., Ferreira, A., Rajamony, R., Rubio, J. (2014, July). Updated Performance Comparison of Virtual Machinesand Linux Containers. p. 3, 4. Austin, Texas. IBM Research Division. Austin Research Laboratory.