An Investigation of Containerization Solutions

Oscar Almqvist
Eric Vickström

April 2021

1 Introduction

Not all machines are configured in the same way, which means running the same programming code yields different behaviour. The idea of containerization is to package programming code, configurations, and dependencies to be easily deployed in any environment [1]. Containerization is a popular technology used in the DevOps pipeline. With the help of containerized solutions, you are able to guarantee the same configurations for both your testing and production environment [2]. This essay aims to investigate state-of-the-art containerization solutions we have today, with the goal of finding their respective strengths, weaknesses, and compare them head-to-head.

2 Containers

Desmond [1] presents his perception of a single container, which is a piece of software abstracted from the operating system. The software has a particular state. The context of the state contains critical information such as the software code, system libraries and system tools. The state of a container is stored in an image, and should be easily transferable.

The Open Container Initiative (abbreviated OCI) is an industry-standard set up in 2015 by, among others, Docker, to normalise the developer interface of how images/containers are created and accessed [3]. Naturally, Docker follows this, but also Podman, which we will go into further detail later; The OCI to this date consists of two separate specifications called the Runtime Specification and the Image Specification [3].

2.1 Relation to Virtualization

A natural question that arises with containers is how it differs from virtual machines. Do both technologies use some form of virtualization? The paper [4] discusses the differences. According to them an operating system (OS) needs access to the hardware. To be able to virtualize, the underlying OS needs to route hardware request from the virtualized OS via a hypervisor. So, for a virtual machine, a complete installation of the OS needs to be conducted. This includes the kernel, which is the central mechanism in an operating system. The authors believe that the main benefit of is the flexibility of running any operating system. Container-based virtualization shares the same kernel as the system, so containers run multiple instances of an OS without duplicating functionality. So containers use fewer resources, with the sacrifice of not running a Linux container on a Window machine.

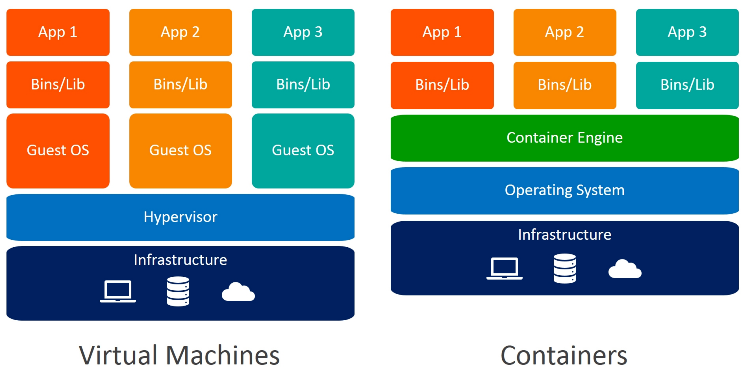

Figure 1: Containerisation vs Virtualisation [1]

2.2 Containers and DevOps

DevOps encompasses a myriad of practices as stated in [2], but two cornerstones are automation and continuous deployment. Containers allow an application to be stored and shared across different environments while still being able to execute without a problem. Containerizing your application thus gives you the possibility to scale it and have consistency across your application users.

3 State-of-the-Art Containers

3.1 Open Container Initiative

According to the OCI [5], the goal of a container is that it should encapsulate a piece of software and its dependencies in a portable way. OCI has defined 5 principles of which the organization believes what a standard container should be. For the first principle, the containers should have a set of operations to setup themselves and prepare the file system. The second principle is that the container should be content-agnostic, which means that regardless of the encapsulated software, each container should be created similarly. The third

[1] Image credit: https://www.weave.works/blog/a-practical-guide-to-choosing-between-docker-containers-and-vms

is infrastructure agnostic; the container should be able to run on any OCI supported infrastructure, regardless of which type of machine it is. The fourth is automation; simply by achieving principle two, you are able to fully automate the process of setting up the container. The fifth and last point is industrial-grade delivery, which is fulfilled by achieving the first four points.

3.1.1 Specifications

If we dive into the documentation of The Image Specification by OCI [6], they state several requirements for a container image. It should consist of a manifest, layout, a set of file system layer and a configuration file. The objective of an image is to help create inner tools for building, transporting and preparing a container. Given the layout, one should be able to create a Runtime Specification bundle (used in the execution of a container). By first, navigating to the manifest, which includes the configuration file and references to the components of the container. The components are defined as layers, and each layer is a set of file system instructions to add, change or delete files. Along with each reference to a layer file, a one-way hash and the size of the content should be stored with it. The hash can be seen as an ID to guarantee that it is the expected set of instructions. The configuration file consists of an ordered set of layer hashes that, if applied, create a complete file system for the container. To guarantee the authenticity of an image, you hash the configuration file as well.

The Runtime Specification by OCI [7] aims to standardize the configuration, execution environment and the life cycle of a single container. The configuration file should contain required metadata to implement standard operations against containers. Examples of standard operations are the process of starting, stopping and deleting a particular container. These operations are stored in the image bundle. The life-cycle of a container is a set of instructions that call functionality found in the bundle, from the creation of the container to the destruction of it. An OS is fundamentally different, and therefore to fulfil the requirements of the specifications, each operating system has provided a platform-specific specification alongside the original one. As of now, there are platform-specific implementations for Windows, Linux and Solaris.

3.2 Containerization Technologies

3.2.1 Docker

Docker is one of the more well-known containerization solutions. The Docker platform [8] not only provides tools to manage containers in a system but also the possibility to manage the images from which the containers are created from[2]. Additionally, Docker is OCI compliant and utilizes client-server architecture. To administrate containers, a user communicates via the docker client. Through the client, commands are sent to the server called the Docker daemon. The daemon is the administrator and manages Docker objects such as the images and the

[2]This is called a Docker registry.

containers. It also monitors the network traffic into the containers. Docker provides a large public registry of Docker images. Via the registry, anyone can download container images.

3.2.2 LXC/LXD

Linux Containers [9], abbreviated as LXC, as the name suggests is specifically for running only on the Linux operating system. These can be created and modified using commands such as `lxc-start`, `lxc-stop`, and more. There exists primarily two different LXC containers: unprivileged and privileged. Unprivileged containers are more restricted in terms of what operations can be performed, but they are the safest from a security point of view. The privileged containers are running as root. Similar to how Docker has a registry where you can download remote images, LXD builds on top of LXC, and it fulfils the need of managing containers and more [10].

LXD is construed by a privileged daemon which is then used to communicate over a REST API. This means that it follows a client-server architecture. Compared to LXC, LXD offers more advanced features as management and control of resources such as network, storage and external devices. As described by Linux Containers themselves, LXD is seen as the modern predecessor of LCX. [10]

3.2.3 Podman

According to [11] [12], Podman is an open-source containerization tool for Linux containers. It is based on the OCI, which makes it flexible being used with similar container solutions. What distinguishes Podman from, for instance, Docker is the fact that it is a daemon-less container manager. This results in Podman interacting directly with all the parts of the containerization context through the runC container runtime. The consequences of this are, for instance, increased security, reliability and lower resource utilization during idle state. Podman also supports running pods; a group of containers sharing resources such as networking and memory.

4 Comparison

There are many aspects to take into account when evaluating containerization solutions from a security perspective. For instance, as Combe, Martin, and Di Pietro [13] describes in their article regarding security aspects of Docker, one has to consider the actual image running. The distribution of images is also considered by the authors, which is something that transcends all the platforms where one can share container images with one another. One of the more decisive differences between Docker and the others is that it can be run on several operating systems than just Linux. If the person in question uses Windows containers for their infrastructure, the choice between which technology to use is easy.

An attribute that makes Podman unique is the daemon-less architecture compared to Docker's and LXD's client-server architecture. For Linux distributions, both Docker and Podman make use of runC to run their respective containers. However, Docker has taken a direction of its own and built upon the original runC. The administrator needs to communicate via the daemon. There are benefits with this, such as a uniformed command-line interface regardless of the operating system. However, as mentioned by [11], each container is dependent on the daemon. Which means that it has a single point of failure. If the daemon is unresponsive, their administrator wouldn't be able to communicate with the containers. According to [14] Podman don't require root privileges, which adds a layer of protection. Docker has answered by recently adding their own rootless setting. The benefits of being rootless are if the container runs on a non-administrator user and it gets compromised, the attacker would have limited privileges to attack the whole system. Podman [15] states that being root-less has its shortcomings, for example, not being able to bind ports below 1024. As stated in [8], Podman has the ability to run containers in pods, which you would need Kubernetes to do if running Docker.

5 Conclusion

The landscape of containers continues to evolve, and standards like the OCI allows for an improved developer experience. Choosing a certain type of container solution depends on your needs and what type of container it is. However, many are interchangeable, such as Docker and Podman, because they both comply with the OCI. Altogether, containers are a building block of DevOps, and containers as a concept are something one should have knowledge of.

References

[1] Ian Desmond. *What is containerization in DevOps*. Mar. 3, 2021. URL: https://www.liquidweb.com/kb/what-is-containerization-in-devops/ (visited on 04/20/2021).

[2] *Understanding the Role of Containers in DevOps*. Sept. 1, 2020. URL: https://www.xenonstack.com/insights/containers-in-devops/ (visited on 04/23/2021).

[3] *OCI - Overview*. OCI. URL: https://opencontainers.org/about/overview/ (visited on 04/20/2021).

[4] Sachchidanand Singh and Nirmala Singh. "Containers & Docker: Emerging roles & future of Cloud technology". In: *2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*. IEEE. 2016, pp. 804–807.

[5] *The 5 principles of Standard Containers*. OCI. URL: https://github.com/opencontainers/runtime-spec/blob/master/principles.md (visited on 04/20/2021).

[6] *Image Format Specification*. OCI. URL: https://github.com/opencontainers/image-spec/blob/master/spec.md (visited on 04/20/2021).

[7] *Open Container Initiative Runtime Specification*. OCI. URL: https://github.com/opencontainers/runtime-spec/blob/master/spec.md (visited on 04/20/2021).

[8] *Docker overview*. URL: https://docs.docker.com/get-started/overview/ (visited on 04/23/2021).

[9] *LXC - Getting started*. URL: https://linuxcontainers.org/lxc/getting-started/ (visited on 04/23/2021).

[10] *What's LXD*. URL: https://linuxcontainers.org/lxd/introduction/ (visited on 04/23/2021).

[11] *Replacing Docker with Podman - Power of Podman*. URL: https://cloudnweb.dev/2019/06/replacing-docker-with-podman-power-of-podman/ (visited on 04/20/2021).

[12] *Podman: A tool for managing OCI containers and pods*. URL: https://github.com/containers/podman/blob/master/README.md (visited on 04/23/2021).

[13] T. Combe, A. Martin, and R. Di Pietro. "To Docker or Not to Docker: A Security Perspective". In: *IEEE Cloud Computing* 3.5 (2016), pp. 54–62. DOI: 10.1109/MCC.2016.100.

[14] Marc Merzinger. *Why Podman is worth a look*. URL: https://www.ti8m.com/blog/Why-Podman-is-worth-a-look-.html (visited on 04/20/2021).

[15] *Shortcomings of Rootless Podman*. OCI. URL: https://github.com/containers/podman/blob/master/rootless.md (visited on 04/20/2021).