Container Comparison

Author: Stefan Alfeis, stefan.alfeis@quant-x-sec.com
Translation: Jeremias Bogomolec, jeremias.@quant-x-sec.com

This whitepaper will deal with the comparison of multiple container systems. Due to the multitude and variety of these systems, this document will only focus on Docker, Kubernetes and Podman. Docker serves as the foundational container software. It was established first and has inspired a large number of programs. Kubernetes and Podman are both administration programs for containers. Their purpose is to facilitate the handling of multiple containers. Below, the variants will each be introduced briefly along with their capabilities.

Docker

Ever since the development of Docker, a lot has happened in the environment of containers. Docker has made it possible to make applications portable, without having to add one's own operating system (OS). Previously that had been the case whenever one created an application within a virtual machine (VM). Through the omission of the OS in Docker, the container has on one hand gotten smaller when compared to a VM, but on the other hand a container is available much faster than a VM, since it does not need to wait for the OS to be running. Similarly to how the VM needs a hypervisor to be started, Docker requires a Docker Engine to get started. Containers can be started and stopped about as quickly as a process. Every container image is capable of harboring its own application without being connected to the host. This raises the portability immensely. The following image serves the purpose of visualizing what I just described:

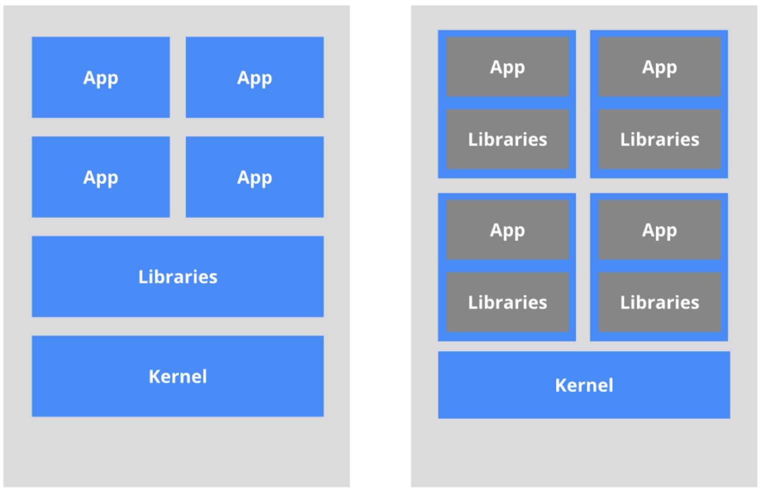

Image: Comparison host to container, Source: Kubernetes.io

Kubernetes

Kubernetes is a container-focused management environment. It is designed to help facilitate the deployment of containers, regardless of whether that deployment is supposed to occur locally or in a cloud. On a fundamental level, Kubernetes was developed in order to improve the

communication between different nodes within a cloud. In Kubernetes the word "pod" also came up for the first time.

Within a pod there are one or multiple containers, which share specific resources and do not run completely isolated from one another. The applications of a pod can however possess specific isolations, depending on their configuration. So Kubernetes arranges the logical units that an application is composed of to ease the processes of administration and identification. However Kubernetes is not just an orchestrating system:

It allows us to break out of the exceptionally linear workflow composition of A -> B -> C and to use independently constructible control processes. It should not matter how one gets from A to C. The desired state should be continuously striven for by the entire orchestrated system.

Podman

The third variant is Podman. Just like the name implies, Pods are an integral component of Podman. Podman is founded on Kubernetes, yet without being limited to only being usable exclusively within a Kubernetes environment. Every pod in Podman possesses an Infra-Container for the administration of certain resources like Namespaces or network ports.
Furthermore, Podman has a process "conmon", for monitoring the containers inside of a pod. One of the biggest advantages of Podman is that the containers are being executed without root permissions. The commands within a container are root-controlled, but the user does not need root permissions to initiate the container. This way, the user is protected, since an attacker would be able to assail the container, but not the host.
Podman also records the actions of a user with sudo-permissions using the auditd-System. With Docker-Socket, this is not the case. As the final point, I'd like to mention how Podman handles the management of containers. When using Docker, the Command Line Interface (CLI) sends all commands to a daemon, and that daemon then distributes them to registry, images, containers and kernel. The caveat here: If the daemon encounters an error, Docker stops working. Podman uses "runC" for this and omits the daemon entirely. RunC is responsible for the execution of a container-process and implements the runtime-specifications. The constitution of Podman's structure can be seen in the image below:

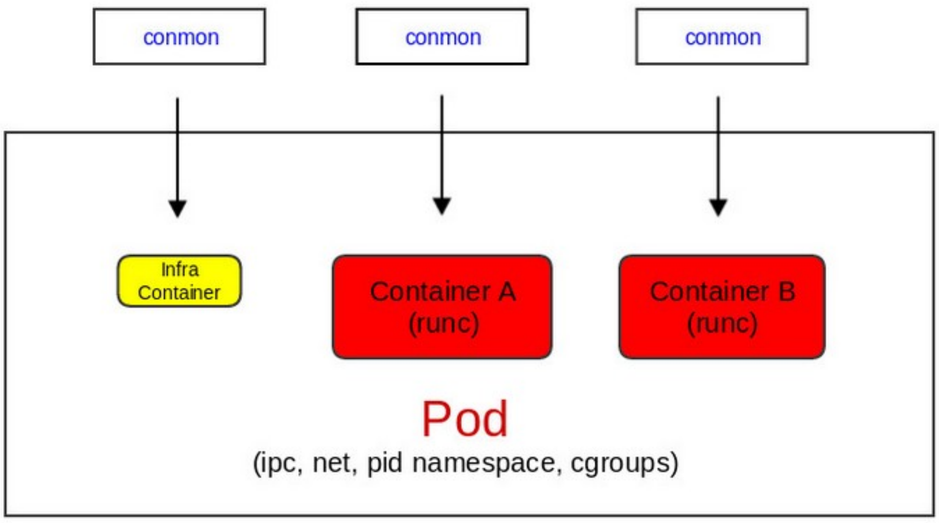

Image: Comparison host to container, Source: netways.de

Conclusion

All three variants have their benefits and disadvantages, but Podman still offers the best concept, since the detachment from the host provides a very high level of security. An attacker can indeed attack the pod or container in itself, but the host stays safe. Thanks to the swift portability, the pod can be restored very quickly without having to configure the host in the case of a security incident.