1 Synopsis

In this article we aim to describes a high-level overview of Containerization, with emphasis on Docker (which has been one the learning subjects in our curriculum). First we will analyze Containerization technologies from a historical perspective, with emphasis on the problems issues that the different technologies have solved. Along with the motivation for further improving upon these technologies, up until Docker. Which is arguably considered state of the art within Containerization tools in the modern day development-operations paradigm.

We then compare Virtualization and Containerization from a high-level perspective, along with the best use cases for each technique.

2 History and origin of Containerization

Containerization has been around for a long time. Chroot first came out in 1979, and gave developers the ability to change the root directory of a process. This feature enables developers to encapsulate programs in a contained file system, which is isolated from the original file system. This idea was expanded upon, when Jails came out in 2000.

Jails provides a modified runtime environment, hereby the name Jails. [1]These processes are "jailed" or isolated from other processes on the host platform. Jails was the first attempt at creating a software that creates independent "mini systems" all sharing the same kernel. This is the main idea of Containerization.

Solaris Containers were developed by Sun Microsystems, the original developer of the Java Programming language. This provided a better way to isolate processes, but these Containerization methods still required a lot of configuration and maintenance.

Later Linux containers were introduced which was a software very similar to since it provides an OS level Virtualization technology, which can be used to either sandbox applications or to emulate an entirely new host. Docker extends upon this concept as it provides a high-level api with a single lightweight Virtualization solution to run isolated processes.

This concept of Linux Containerization is also referred to as virtual environments, which are different from Virtual Machines. In virtual environments there is no preloaded emulation manager software as there is in a Virtual Machine. This can help us optimize performance, because each isolated instance only packages the needed applications and respective dependencies. Where Virtual Machines package an entire operating system, and the underlying infrastructure.

[2]In 2013 Docker was introduced which solved a lot of the issues with regards to deploying applications on Virtual Machines and Virtual Environments as infrastructure providers.

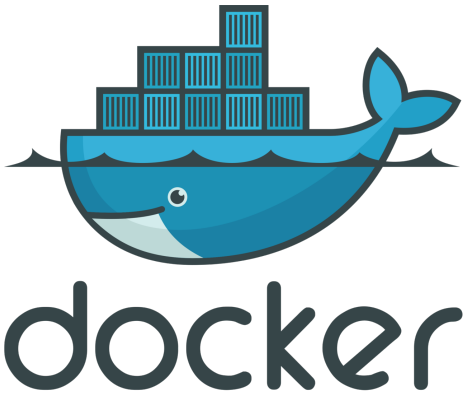

Figure 1: Docker logo

The etymological origin of Docker as well as the design of the logo, is an analogy of the shipping and container industry. Similar to how shipping solves the logistical issues of manually loading, unloading, protecting fragile merchandise and orchestrating different shipments between several countries and continents.

Docker solves similar issues present in the software industry with regards to manual development of containers, difficulty of isolating processes and different platforms requiring specific configurations, and dependencies.

3 Containerization in contrast to Virtualization

What is the difference between Virtualization and containers?

When comparing Virtualization to Containerization, Virtualization enables the user to run multiple operating systems on the hardware infrastructure of a single physical server. Virtualization is typically used in instances, where we need full functionality from the underlying operating system. This advancement in Virtualization is often seen in state of the art web applications.

When comparing Containerization to Virtualization, Containerization provides abstraction on an operating-system level and limits the resources available to the machine as per required, to run the application in the container. Virtualization is a hardware-level abstraction where we ship the entire operating system, meaning that it has its own kernel implementation. Containers share the same kernel. This also means that Virtualization provides full isolation and increased security, while containers are process level isolated which essentially provides a less secure environment.

When we think of Virtualization, the most common practise is to use Virtual Machines. As mentioned above, this is a hardware level abstraction, meaning it lays between the operating system and the physical hardware layer.

The Virtual Machine consists of the actual Virtual Machine along with a hypervisor, this is the management software that allows an Operating System to run multiple Virtual Machines sharing the same physical resources.

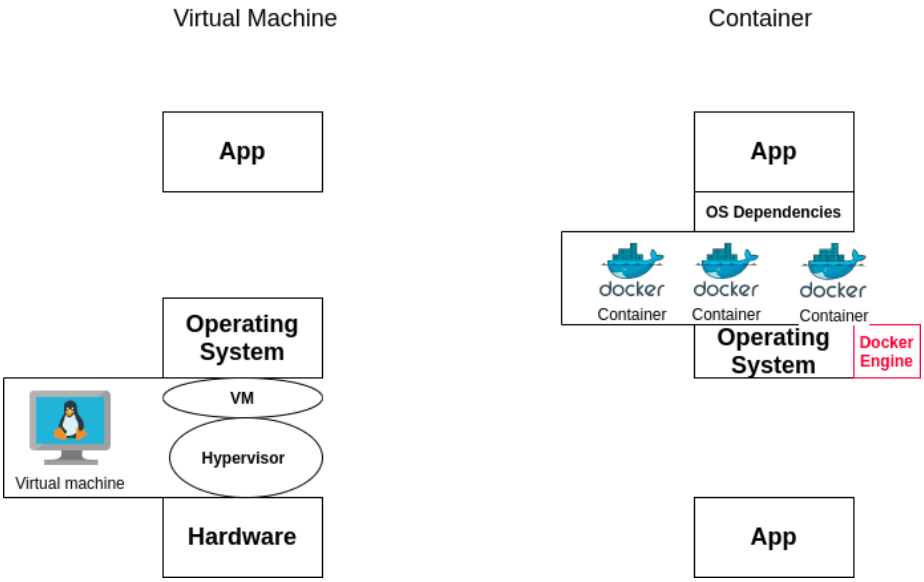

Figure 2: Diagram illustrating the differences between Containerization and Virtualization

Illustrated above is the abstraction between Operating System and application. By using Docker the Operating System can run several applications in containers, if utilizing Virtual Machines. The difference is, In theory these practises can be used alongside each other, as Docker containers can also run on a Virtual Machine. The Docker Engine is the technology that spins up Docker containers, and it only requires an operating system to function. It does not take the physical hardware specifications into account. Whereas a Virtual Machine, is an abstraction in between the operating system, and the physical hardware.

What is the best use case for each respective practise? In regards to cloud technology and dev-ops, when do we want to use Virtualization and when do we want to use Containerization?

4 Containerization pro's and con's

In the modern day application development landscape, Containerization can be a powerful technique for optimizing deployment of applications . But there are also situations, where you might not want to use it.

Cons of using Docker: Containers typically rely on a base image. This could for example be a nodeJS image or a python image, if we were trying to run a web application. It could also be a centos or ubuntu image if we wanted an operating system as the base image. Depending on the situation, base images can contain vulnerabilities because they are maintained by external parties. [3] In some use cases, there are special compliance requirements that require you to manually manage the infrastructure. In recent years, the global political landscape has had an increased focus on data compliance and security measures when storing user data. Since the GDPR (General Data Protection Rights) agreement, made

by the European Union in 2016, along with multiple other regulations, has ensured that certain industries have special regulatory requirements, in terms of hardware storage and encryption. When we abstract the physical layer using Containerization and Virtualization, there are some limitations to the hardware, this means we can't control the hardware storage.

Cons of not using Docker: Without Docker we would have to manually set up a development environment for our application to function. This would mean installing and requiring dependencies, setting up configuration files, managing infrastructure etc. This would result in a necessity to set up replica environments for each new application This equates to less scalability, decreased security and a longer more complicated setup process giving higher error and misconfiguration margins

To elaborate on the first issue, a similar issue arises when we utilize multiple environments for our application. We might have multiple different environments, eg: development, staging, production etc. These would each require individual maintenance and configuration, by the time we move from development or staging into a production environment, the dependencies may be outdated, this could result in ending up with a different version than our intended environment. Unlike Docker that provides us with an easy way to spin up, stop, and restart containers. We have no simple way of running these commands with Virtual Machines as we would have to start and stop the actual Virtual Machines, not just the application environment. Without Docker we are less flexible in terms of moving and reproducing our environment. Setting up and configuring environments poses some obstacles. If we migrate to a new cloud provider or a new platform, it might be a slow and inefficient process. Because Docker is platform agnostic, it provides us a simple interface to connect to the underlying platform.

5 Conclusion

As elaborated on throughout the article, specific use-cases becomes clear whether to implement your software over Containerization or Virtualization. Each of the respective paradigms each provides specific circumstances to run software, and the conditions determines whether contained or virtual environments is appropriate.

If your hosted processes requires hardware access, then a Virtual Environment would be the obvious choice. Other relevant considerations could include concerns in regard to interactions, management and deployment. Even though the popularity of Containerization is abundantly clear these days, and is the de facto for more general software implementations, it does not always provide the same utilities that Virtualization does.

References

[1] Ricardo Vardasca Carlos Antunesa. *Performance of Jails versus Virtualization for Cloud Computing Solutions*, 2014.

[2] Jacopo Soldani Claus Pahl and Pooyan Jamshidi. *Cloud Container Technologies:a State-of-the-Art Revie*, May 2017.

[3] Gerald Fredrick Lofstead, Ivo Jimenez, Carlos Maltzahn, Adam Moody, Kathryn Mohror, and Remzi Arpaci-Dusseau. The role of container technology in reproducible computer systems research.